VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

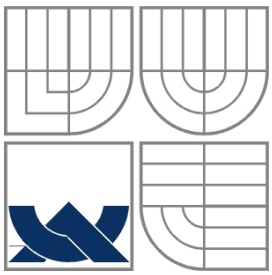
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ALGORITMY SOUBĚŽNÉHO TECHNICKÉHO A PROGRAMOVÉHO NÁVRHU

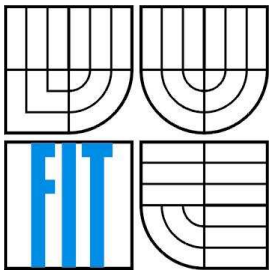
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE JAN VLACH
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ALGORITMY SOUBĚŽNÉHO TECHNICKÉHO A PROGRAMOVÉHO NÁVRHU

HARDWARE-SOFTWARE CODESIGN ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JAN VLACH
AUTHOR

VEDOUCÍ PRÁCE — Dr. Ing. OTTO FUČÍK
SUPERVISOR

BRNO 2007

Abstrakt

Tento diplomový projekt se zabývá souběžným návrhem programového a technického vybavení vestavěných systémů. Zahrnuje jednak obecný popis celého tohoto procesu , jednak je tento postup ilustrován na návrhu, simulaci a implementaci FIR filtru. Je zde obsažen také popis návrhového programu Polis a simulačního systému Ptolemy. Závěr projektu je věnován generování simulačních modelů v jazyce VHDL včetně následné syntézy.

Klíčová slova

Souběžný návrh technického a programového vybavení, souběžná simulace technického a programového vybavení, specifikační jazyk Esterel, program Polis, simulační systém Ptolemy, VHDL

Abstract

This master’s thesis deals with a parallel design of the program and a technical equipment of embedded systems. It involves both a general description of the whole process and an illustration of the design, a simulation and implementation of the FIR filter. It also includes a description of the proposed program Polis and the simulation system Ptolemy. The conclusion of the project is devoted to a generation of simulation models in VHDL language incl. a subsequent synthesis.

Keywords

Hardware-software codesign, hardware-software cosimulation, specification language Esterel, program Polis, simulation system Ptolemy, VHDL

Citace

Vlach Jan : Algoritmy souběžného technického a programového návrhu. Brno, 2007,
diplomová práce, FIT VUT v Brně.

Algoritmy souběžného technického a programového návrhu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením. Dr. Ing. Otto Fučíka. Další informace mi poskytli Dr. Ing. Petr Peringer a Luciano Lavagno z Politecnico di Torino, který je jedním z autorů programu Polis.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Jan Vlach

.....

31. července 2007

Poděkování

Děkuji vedoucímu práce Dr. Ing. Otto Fučíkovi, Dr. Ing. Petru Peringerovi, Doc. Ing.Josefu Schwarzovi, CSc., jakož i panu Lucianovi Lavagnovi za neocenitelnou pomoc při vypracování tohoto diplomového projektu.

Obsah

1 Úvod

Vestavěné systémy bývají implementovány jako smíšené systémy (mixed hardware/software systems), tj. systémy, které obsahují jak technické, tak programové vybavení. Běžným postupem při návrhu vestavěných systémů je specifikace a návrh hardwaru odděleně od specifikace a návrhu softwaru. Tento přístup však může mít za následek jednak nekompatibilitu mezi hardwarovou s softwarovou částí systému, jednak předem stanovené hardware/software rozdělení obvykle vede k suboptimálním výsledným návrhům.

Byly proto vytvořeny metody souběžného návrhu technického a programového vybavení vestavěných systémů, které se snaží výše zmíněné nedostatky odstranit. Takový přístup vyžaduje použití **jednotné reprezentace** (unified hardware/software representation), která popisuje pouze funkčnost určité jednotky systému, není však specifikováno, zda má být tato jednotka implementována v hardwaru nebo v softwaru.

Pro souběžný návrh vestavěných systémů s využitím jednotné reprezentace bylo vyvinuto několik metodologií pro jejich specifikaci, návrh, automatickou syntézu a validaci. Jedna z těchto metodologií je vyvíjena na University of California, Berkeley, kde bylo pro tyto účely vytvořeno návrhové prostředí POLIS.

2 Návrhové prostředí POLIS

POLIS je softwarový program pro souběžný návrh technického a programového vybavení pro vestavěné systémy (hardware/software codesign of embedded systems). Termínem **hardware/software** jsou označovány systémy, jejichž případná implementace bude obsahovat hardwarovou část a softwarovou část, přičemž softwarová část musí obsahovat hardware, na kterém daný software poběží. **Codesign** je proces vytváření smíšeného (tj. hardware/software) systému. Tento proces zahrnuje specifikaci, syntézu, odhad parametrů (estimation) a verifikaci. **Vestavěný systém** (embedded system) je neformálně definován jako kolekce programovatelných částí, které jsou obklopeny obvody ASIC a dalšími standardními komponentami, které průběžně interagují s okolním prostředím prostřednictvím senzorů a akčních členů (actuators). Programovatelné části zahrnují mikrokontroléry a procesory DSP.

Předmětem návrhu je hierarchický netlist (hierarchical netlist), jehož listy tvoří rozšířené konečné stavové stroje zahrnující reaktivní řízení a datovou cestu. Takový konečný stavový stroj se označuje zkratkou CFSM (Codesign Finite State Machine). Střední úrovně hierarchického netlistu se nazývají netlisty (netlists). Netlist i CFSM se souhrnně označují jako **moduly** (modules).

CSFM podobně jako FSM transformuje množinu vstupů na množinu výstupů s využitím konečného počtu vnitřních stavů. Rozdíl mezi CFSM a FSM je v tom, že synchronní model komunikace mezi jednotlivými FSM je nahrazen nenulovým, konečným reakčním časem (reaction time), jehož velikost však není nijak omezena. Takový výpočetní model se označuje jako globálně asynchronní a lokálně synchronní. Celý navrhovaný systém je reprezentován jako síť jednotlivých CFSM, kde každý CFSM představuje určitou modelovanou jednotku systému. Každý CFSM představuje jednotnou reprezentaci jemu příslušející jednotky systému, protože není určeno, zda bude tato jednotka implementována v hardwaru nebo v softwaru. Hardwarová i softwarová implementace CFSM provádějí stejný výpočet pro každý přechod CFSM, liší se však ve zpožďovacích vlastnostech (delay characteristics). V případě synchronní hardwarové implementace CFSM je možné provést přechod CFSM za dobu jednoho hodinového cyklu, zatímco v případě softwarové implementace CFSM bude tento přechod vyžadovat více než 1 hodinový cyklus.

Každý přechod CFSM je chápán jako atomická operace. Před každým provedením přechodu CFSM je zachycen stav systému. Provedením přechodu se aktualizuje vnitřní stav a výstupy CFSM, který provedl přechod, a výsledek tohoto přechodu je šířen do dalších CFSM a do prostředí (environment) systému.

Interakce mezi jednotlivými CFSM je asynchronní . Doba provedení jakéhokoliv přechodu CFSM je tedy předem neznámá. Jednotlivé CFSM mezi sebou nekomunikují prostřednictvím sdílených proměnných (shared variables), ale prostřednictvím událostí (events).

Termínem **signál** (signal) se označuje nosič událostí, které jsou použity jako základní synchronizační a komunikační mechanismus mezi jednotlivými CFSM. Událost může být emitována kterýmkoliv CSFM, který má definovány výstupní signály, nebo prostředím systému. Emise události může být detekována jinými CFSM. Každá emise události může být určitým CFSM detekována nejvýše jednou. K detekci události dochází s určitým nenulovým zpožděním od okamžiku emise události, které závisí na více faktorech, mezi něž patří např. aktuální hardware/software rozdělení nebo zvolený typ plánování (scheduling policy). Každý CFSM, který detekuje nějakou událost, si vytvoří svou vlastní kopii této události.

Signály mohou nést řídicí informace, datové informace nebo obojí. Signály, nesoucí pouze řídicí informace, se nazývají **čisté signály** (pure signals), zatímco signály, nesoucí pouze datové informace, se označují jako **čisté hodnoty** (pure values).

V systému POLIS není poskytnuta možnost použít zachytávání hodnot ve vyrovnávacích pamětech

(buffering). Pokud tedy bude CFSM, který emituje událost, rychlejší než CFSM, který má tuto událost přijmout, může být tato událost přepsána jinou událostí emitovanou jiným CFSM. Tento jev se také nazývá
ztrácení (losing) událostí. Aby se zabránilo ztrátě události, je třeba provést určitá opatření. Jednou z možností je vytvořit explicitní handshake mechanismus mezi jednotlivými CFSM ve formě dvojic nebo množin signálů. Jinou možností je implementovat CFSM, který přijímá událost, v hardwaru nebo implementovat oba CFSM (vysílající i přijímající) v softwaru a použít plánovač typu round-robin, který bude oba dva CFSM provádět stejnou rychlostí.

Kroky návrhu a implementace systému s použitím programu POLIS jsou následující:

Mimo program POLIS:

1. Zápis počáteční specifikace vestavěného systému ve vysokoúrovňovém jazyce (high level language) (je možné použít například jazyk ESTEREL, Lustre nebo Signal), ve kterém se specifikuje každý modul navrhovaného systému. V současné době je pro program POLIS doporučeno použít jako specifikační jazyk ESTEREL. Sémantika souběžného sestavování modulů je odlišná v jazyce ESTEREL a v programu POLIS. V jazyce ESTEREL jsou souběžné moduly prováděny synchronně, což znamená, že výpočet trvá nulovou dobu a výsledek souběžného skládání je nezávislý na fyzické implementaci (typ plánování, atd.). V programu POLIS jsou souběžné moduly prováděny asynchronně, v závislosti na vybraném mechanismu plánování. To však platí pouze pro moduly, které jsou určeny pro implementaci v softwaru. Moduly určené pro implementaci v hardwaru jsou prováděny souběžně i v programu POLIS.

2. Specifikované moduly se přeloží do formátu SHIFT (Software Hardware Interface FormaT) pomocí příkazu strl2shift (ten zavolá překladač jazyka ESTEREL, který přeloží zdrojový soubor ve formátu strl do intermediárního formátu oc. Poté je zavolán překladař oc2shift, který soubor ve formátu oc přeloží do formátu SHIFT). Formát SHIFT specifikuje topologii sítě interagujících CFSM a definuje chování jednotlivých CFSM. Je zde jeden CFSM pro každý specifikovaný modul.

V programu POLIS:

1. Návrh se načte do programu POLIS.

2. Vytvoří se vnitřní grafová reprezentace.

3. Nad tímto grafem se provede optimalizace.

4. Vybere se cílový mikroprocesor, který bude použit pro běh softwaru ve výsledné implementaci.

5. Spustí se odhadovací nástroje (estimation tools), které nám dají představu o velikosti a rychlosti výsledného softwaru.

6. V jazyce C se pro každý softwarový modul vygeneruje kód (*.c).

7. V jazyce pro simulační systém Ptolemy se vygeneruje kód pro každý modul (*.pl).

8. Opustí se prostředí POLIS.

Mimo program POLIS:

1. Použijí se soubory makefile systému POLIS pro vytvoření dostatečných informací pro systém Ptolemy, ve kterém se bude simulovat celý návrh.

2. Spustí se systém Ptolemy. Zde se zkontroluje vzájemné propojení modulů, nastaví se parametry,

spustí se simulace, ve které se ověří, zda se návrh chová tak, jak se očekává, případně se vytvoří jiné hardware/software rozdělení (hardware/software partition) návrhu, aby se zlepšil výkon.

3. Jakmile je systém verifikován touto smíšenou (hardware/software) simulací, vytvoří se skutečná implementace návrhu.

V programu POLIS:

1. Implementace návrhu, vytvořená v předchozím kroku, se načte do programu POLIS (zde bude obecně zapotřebí pomocný soubor, ve kterém budou definována vzájemná propojení modulů).

2. Pro každý modul se určí, zda bude implementován v hardwaru nebo v softwaru.

3. Pro hardwarovou část:
a) Vnitřní formát se přeloží do hardwarového formátu.
b) Tato reprezentace se optimalizuje.
c) Hardware se vypíše ve formátu netlistu.

4. Pro softwarovou část:
a) Vytvoří se vnitřní graf reprezentující software.
b) Tento graf se optimalizuje.
c) Vybere se cílový mikroprocesor.
d) Spustí se odhadovací nástroje, které nám dají představu o velikosti a rychlosti výsledného softwaru. Volba předaná odhadovacímu nástroji specifikuje vybraný mikroprocesor.
e) Vygeneruje se kód v jazyce C pro každý softwarový modul.
f) Vygeneruje se operační systém (také soubor v jazyce C).

5. Opustí se prostředí POLIS.

Mimo program POLIS:

1. Vybere se hardware (např. FPGA Xilinx).

2. Soubory s netlisty se přeloží do hardwaru (např. použitím softwarových nástrojů Xilinx se namapuje vygenerovaný netlist do netlistu, který je vhodný pro danou architekturu a potom se tento netlist nahraje do FPGA).

3. FPGA se zasune do desky.

4. S pomocí softwarových a hardwarových nástrojů dodaných s mikroprocesorem (nebo spíše vestavěných mikrokontrolérem, který obsahuje mikroprocesor a paměť) se uloží soubory modulů v jazyce C a soubor s operačním systémem v jazyce C do mikrokontroléru.

5. Mikrokontrolér se zasune do desky.

3 Systém Ptolemy

Pro souběžnou simulaci (kosimulaci) technického a programového vybavení (hardware/software co-simulation) je možno použít mnoho systémů. Mezi ně patří například komerční simulátory VHDL a systém Ptolemy.V případě použití systému Ptolemy je pro systémy, jejichž popis je vygenerován v programu POLIS, nutné použít verzi Ptolemy Classic, nikoliv novější verzi Ptolemy II.

Základem systému Ptolemy je kompaktní softwarová infrastruktura, nad kterou je možno vybudovat specializovaná návrhová prostředí, která se nazývají domény. Tato softwarová infrastruktura se nazývá jádro systému Ptolemy (Ptolemy kernel). Je tvořeno skupinou definic C++ tříd. Domény jsou definovány vytvořením nových C++ tříd, odvozených ze základních tříd v jádře. Použití objektivně orientované softwarové technologie umožňuje, aby jedna doména interagovala s jinou doménou bez znalosti vlastností nebo sémantiky té druhé domény. Použijí-li se tedy různé domény, může tým návrhářů modelovat každý podsystém složitého heterogenního systému přirozeným a efektivním způsobem. Tyto různé podsystémy mohou být do sebe vnořeny a tvořit tak strom podsystémů. Ptolemy, díky tomu, že podporuje heterogenitu, poskytuje "výzkumnou laboratoř" pro testování a zkoumání metodologií návrhu, které podporují mnoho stylů návrhu a mnoho implementačních technologií.

3.1 Historie systému Ptolemy

Ptolemy je třetí generací softwarového prostředí, jehož počátek spadá do ledna 1990. Je přímým následníkem dvou předchozích generací návrhových prostředí, z nichž první se jmenuje Blosim a druhé Gabriel. Návrhová prostředí Blosim a Gabriel jsou zaměřena na číslicové zpracování signálu (DSP) a obě používají dataflow sémantiku se syntaxí ve formě blokových diagramů pro popis algoritmů. Aby se rozšířila použitelnost i za hranice oblasti číslicového zpracování signálu, jádro systému Ptolemy nemá zabudovánu dataflow sémantiku, ale místo toho poskytuje podporu pro širokou paletu výpočetních modelů, mezi něž patří dataflow, zpracování diskrétních událostí, komunikující sekvenční procesy, výpočetní modely založené na sdílených datových strukturách a konečné stavové stroje. Pro tyto výpočetní modely poskytuje jádro systému Ptolemy směs technik plánování v době překladu (compile time) a v době běhu (run time). Na rozdíl od systémů Blosim a Gabriel poskytuje jádro systému Ptolemy infrastrukturu, kterou je možno rozšířit o nové modely výpočtu bez nutnosti znovu implementovat celý systém. Od roku 1990 bylo uvolněno sedm větších vydání (releases) systému Ptolemy, které jsou číslovány 0.1 až 0.7. Nula v čísle vydání vyjadřuje, že Ptolemy je výzkumný software a ne komerční produkt. Flexibilita systému Ptolemy je zejména důležitá, aby byl umožněn výzkum metodologie návrhu.

3.2 Jádro systému Ptolemy

Typické použití systému Ptolemy zahrnuje spuštění dvou procesů. První proces obsahuje uživatelské rozhraní **vem** a databázi návrhu **oct**, druhý proces obsahuje jádro systému Ptolemy. Alternativně je možné spustit systém Ptolemy bez grafického uživatelského rozhraní jako jeden proces. V tomto případě je použit textový interpret **ptcl** (Ptolemy Tcl), založený na jazyce Tcl (Tool Command

Language).

Jádro systému Ptolemy poskytuje největší podporu pro domény, ve kterých je návrh reprezentován jako síť bloků. Základní třída v jádře systému Ptolemy, která se nazývá Block, reprezentuje objekt v této síti. Jsou také poskytnuty třídy pro vzájemné propojení bloků (třída PortHole), pro přenos dat mezi bloky (třída Geodesic) a pro řízení efektivního "sbírání smetí" (garbage collection) (třída Plasma). Ne všechny domény používají tyto třídy, ale většina současných domén ano, a proto mohou velmi efektivně používat tuto infrastrukturu.

Bloky mohou být hierarchické. Nejnižší úroveň hierarchie je odvozena ze základní třídy jádra, která se jmenuje Star. Hierarchický blok se nazývá Galaxy a reprezentace systému na nejvyšší úrovni (top level) se nazývá Universe.

3.3 Modely výpočtu

Jádro systému Ptolemy nedefinuje žádný model výpočtu. Je ponecháno na konkrétní doméně, aby definovala sémantiku výpočetního modelu. Sémantika domény je definována třídami, které řídí provádění (execution) specifikace. Tyto třídy mohou volat simulátor, generovat kód nebo volat sofistikovaný kompilátor.

3.4 Modely výpočtu založené na toku dat (dataflow models of computation)

Jedna z nejvyspělejších domén, které jsou v současné době zahrnuty do systému Ptolemy, je doména synchronního toku dat (synchronous dataflow (**SDF**)). Tato doména se používá pro zpracování signálů a pro vývoj komunikačních algoritmů. Doména dynamického toku dat (dynamic dataflow domain (**DDF**)) rozšiřuje doménu SDF tak, že umožňuje datově závislý tok řízení. Dataflow sémantiku používá i několik domén pro generování kódu. Tyto domény jsou schopny syntetizovat kód v jazyce C, kód v assembleru pro určité programovatelné DSP procesory nebo kód v jazyce VHDL.

3.5 Modely výpočtu založené na diskrétních událostech (discrete-event models of computation)

Přestože bylo pro systém Ptolemy vyvinuto mnoho simulačních domén s discrete-event sémantikou, v Ptolemy 0.7 se vyskytuje pouze jedna z nich - doména **DE**. Doména DE je generické modelovací prostředí s diskrétními událostmi. Je užitečná pro simulování systémů hromadné obsluhy, komunikujících sítí a hardwarových systémů.

3.6 Synchronní reaktivní modelování

Pro synchronní reaktivní modelování je určena doména **SR**. Jde o softwarovou analogii synchronních číslicových obvodů. Tento model výpočtu je vhodnější než dataflow pro řídicí (control-intensive) aplikace a je efektivnější než DE.

3.7 Smíchání modelů výpočtu

Rozsáhlé systémy jsou často složeny z hardwarových, softwarových a komunikačních podsystémů. Například hardwarový podsystém může zahrnovat různé komponenty, jako jsou zákaznické obvody (custom logic), procesory s různým stupněm programovatelnosti, systolická pole (systolic arrays) a víceprocesorové (multiprocessor) podsystémy. Nástroje podporující každou z těchto komponent jsou obecně různé. Mohou používat dataflow principy, běžné iterační algoritmy, komunikující sekvenční procesy, funkcionální jazyky, konečné stavové stroje nebo teorii a simulaci systémů s diskrétními událostmi.

V systému Ptolemy mohou být domény smíchány a dokonce do sebe zanořeny. Popis systému (system-level description) může tedy obsahovat více podsystémů, které jsou navrženy nebo specifikovány za použití různých stylů.

3.8 Generování kódu

Domény v systému Ptolemy se dělí do dvou skupin. Do první skupiny patří domény pro simulaci, do druhé skupiny potom domény pro generování kódu. V simulačních doménách volá plánovač (scheduler) metody bloků ve specifikaci systému a tyto metody provádějí funkci spojenou s návrhem. V doménách pro generování kódu plánovač také volá metody bloků, ale tyto metody syntetizují kód v nějakém jazyce. To znamená, že generují kód, který má vykonávat nějakou funkci, místo toho, aby vykonaly tuto funkci přímo. Tento mechanismus není závislý na žádném konkrétním jazyku. V systému Ptolemy jsou v současné době zahrnuty generátory kódu pro jazyk C, assembler Motorola 56000 a pro jazyk VHDL.

3.9 Doména DE

Tato doména je určena pro časovou simulaci například komunikačních sítí nebo vysokoúrovňových modelů počítačových architektur. V této doméně každá částice (particle) reprezentuje událost, která odpovídá změně stavu systému. Plánovače v doméně DE zpracovávají události v chronologickém pořadí. Každá částice má přiřazeno **časové razítko** (time stamp), které určuje, kdy (v simulačním

čase) má být zpracována.

Protože události jsou v čase rozprostřeny obecně nepravidelně, jsou všechny akce související s plánováním provedeny v době běhu. Plánovač v doméně DE zpracovává události až do okamžiku dosažení tzv. stop time, což je čas zadaný před spuštěním simulace, který udává okamžik jejího konce.

Plánovač udržuje **globální frontu událostí** (global event queue), ve které jsou všechny částice, nacházející se aktuálně v systému, seřazeny podle svého časového razítka, tj. nejranější nezpracovaná událost se vyskytuje na čele fronty.

V doméně DE jsou v současné době implementovány dva plánovače. Rozdíl mezi nimi spočívá v různém zacházení s globální frontou událostí. Výchozí (default) plánovač domény DE je založen na mechanismu kalendářní fronty (calendar queue). Tento mechanismus zachází s rozsáhlými frontami událostí mnohem méně efektivně než druhý, alternativní plánovač v doméně DE, který seřazený seznam s možností lineárního vyhledávání.

Oba plánovače načítají událost z čela fronty událostí a posílají ji na vstupní porty jejího cílového bloku. Hvězda v doméně DE je provedena (fired), kdykoliv se nová událost vyskytne na některém z jejích vstupů. Před vlastním provedením hvězdy prohledává plánovač frontu událostí, aby zjistil, zda se nevyskytují současné (simultaneous) události na jiných vstupech této hvězdy. Pokud takové události nalezne, načte je také. Při každém provedení může hvězda zkonzumovat všechny současné události na svých vstupech. Jakmile je blok hvězdy proveden, může generovat nějaké výstupní události na své výstupní porty. Tyto události jsou poté umístěny do globální fronty událostí. Poté plánovač načte další událost z čela globální fronty událostí a tento proces se opakuje až do doby, kdy je splněna ukončovací podmínka (stopping condition).

Některé hvězdy v doméně DE jsou pouze generátory událostí, žádné události nekonzumují. Nemohou být proto spuštěny (triggered) vstupními událostmi. Tyto hvězdy jsou nejprve spuštěny pomocí částic, které jsou generovány systémem a umístěny do fronty událostí ještě předtím než je systém spuštěn. Následná provedení těchto hvězd jsou vyžadována hvězdou samotnou, která si sama určí čas, kdy má být znovu provedena. Všechny hvězdy, které mají tuto vlastnost, jsou odvozeny z bázové třídy **RepeatStar**. Ve třídě RepeatStar je vytvořen skrytý vstupní a skrytý výstupní port tak, že jsou oba tyto porty propojeny navzájem. Díky tomuto propojení má hvězda schopnost plánovat své vlastní provedení v požadovaném čase v budoucnu tak, že emituje události s odpovídajícími časovými razítky na svou zpětnovazební smyčku propojující výstupní port se vstupním portem téže hvězdy. Tyto skryté porty se nijak neliší od ostatních portů, s výjimkou toho, že nejsou viditelné v grafickém uživatelském rozhraní.

Jestliže dvě různé hvězdy mají na svých vstupech události s identickými časovými razítky a obě mohou být tedy v následujícím okamžiku provedeny, musí se nějakým způsobem určit, která z nich bude provedena dříve. Běžně se postupuje tak, že je zvolena libovolná z nich. To však může vést na neočekávané výsledky simulace. Je též možné přiřadit pro tuto situaci hvězdám priority, podle kterých se určí, která z hvězd bude provedena dříve.

V DE doméně existuje mnoho hvězd, které na svých výstupních portech produkují události, jejichž časová razítka mají stejnou hodnotu jako je hodnota časových razítek událostí, které se současně vyskytují na jejich vstupních portech. Takové hvězdy mají nulové zpoždění (zero delay). Jestliže je výstupní port takové hvězdy propojen se vstupním portem téže hvězdy, nazývá se toto uspořádání nezpožděná smyčka (delay-free loop). Pokud by se takovéto zapojení vyskytlo v doméně SDF, výsledkem by byl deadlock systému. V případě domény DE však nezpožděná smyčka může potenciálně způsobit neomezený výpočet (unbounded computation). Je proto žádoucí detekovat nezpožděné smyčky v době překladu. Pokud je nezpožděná smyčka detekována, je signalizována chyba.

V DE doméně je pod pojmem čas myšlen simulační čas. Doménu DE je možné použít v kombinaci

s jinými doménami systému Ptolemy. Tyto domény však nemusí mít definován simulační čas. Pro simulaci systému s takovou kombinací domén je možné použít více plánovačů zároveň, z nichž některé využívají simulační čas a některé ne. V rámci jedné simulace se může dokonce vyskytovat i více plánovačů domény DE současně.

4 Kosimulace a hardware/software dělení v systému Ptolemy

Velkým problémem při návrhu vestavěných systémů je velké množství různých variant návrhu (design options), které často vedou na naprosto odlišné hodnoty ceny a výkonu. Program POLIS předpokládá ohodnocení různých hardware/software kompromisů (hardware/software trade-offs) prostřednictvím simulace a ne prostřednictvím matematické analýzy.

4.1 Generování simulačního modelu pro systém Ptolemy

Systém Ptolemy může být použit jak pro funkční (functional), tak pro vysokoúrovňovou (high-level) časovou (timing) simulaci. Funkční simulace může být provedena tak, že se vybere hardwarová implementace pro každý CFSM, protože takové nastavení odpovídá provedení všeho současně v co nejkratším čase. Během vysokoúrovňové časové simulace může návrhář dynamicky (tj. bez nutnosti rekompilovat simulační modely) a hierarchicky (tj. s šířením na nižší hierarchické úrovně) vybrat:

- hardwarovou nebo softwarovou implementaci pro každý CFSM

- typ a frekvenci hodin zdroje, na kterém běží CFSM

- typ plánovače (zda je preemptivní nebo ne) a prioritu každého CFSM (jestliže to vyžaduje metoda plánování).

Potom, co je vygenerován soubor SHIFT pro každý CFSM, je potřeba vygenerovat dva soubory, které budou v systému Ptolemy použity jako simulační modely pro každý CFSM:

1. soubor .pl, který popisuje:
a) vstupy, výstupy a senzory
b) stavové proměnné
c) konstanty, které mají být dynamicky změněny během simulace
d) rozhraní mezi Ptolemy simulátorem a v prostředí POLIS syntetizovaném kódu v jazyce C

2. soubor .c, který popisuje chování CFSM a odhadované počty hodinových cyklů na různých procesorech, které budou potřeba pro provedení daného kódu.

Rozdíl mezi simulací a vlastním prováděním je v tom, že v případě simulace jsou vstupy/výstupy ovládány systémem Ptolemy, zatímco při vlastním provádění jsou ovládány Real-Time operačním systémem (Real-Time Operating System).

Jakmile jsou simulační soubory vytvořeny, musí být přeloženy příkazem make all v adresáři, ve kterém byly vytvořeny (v tomto adresáři program POLIS vytvořil i příslušný makefile).

4.2 Kosimulace v systému Ptolemy

Simulace v systému Ptolemy vyžaduje, aby návrhář vytvořil nejméně dvě hierarchické jednotky - **galaxie** (v terminologii systému Ptolemy). Jednu pro vestavěný systém, který je navrhován a jednu jako simulační testovací sadu. Galaxie systému může být dále dělena do podgalaxií, aby se zvládla složitost návrhu. Testovací galaxie instantizuje galaxii systému a také generátory stimulů a sondy.

Prvním krokem je vytvoření galaxie systému. Ta se vytvoří tak, že se přesuneme do podadresáře ptolemy (ten byl vytvořen programem POLIS) a zadáme příkaz pigi system_name &.

Spustí se uživatelské rozhraní pigi (Ptolemy interactive graphical interface). Prvním krokem, který se musí pro každou vytvořenou galaxii udělat, je, že pro ni vybereme doménu (tím vybereme simulační mechanismus). Pro všechny simulace objektů vytvořených v programu POLIS se používá doména DE. Druhým krokem je vytvoření ikon pro jednotlivé CFSM. Dále je nutné navrhnout galaxii systému propojením instancí hvězd (stars). Propojení se vytvoří nakreslením cesty mezi terminály ikon. Uživatelské
hvězdy mají vždy vstupní terminály nalevo a výstupní terminály napravo. Dalším krokem je editování parametrů instancí hvězd a galaxií. Posledním úkolem fáze zachycení návrhu (design capture phase) je vytvoření ikony pro galaxii systému.

Následuje vytvoření testovací galaxie, pro kterou musí být také zvolena doména DE. Testovací galaxie musí obsahovat právě jednu instanci galaxie navrhovaného systému. Simulovaná galaxie musí být v testovací galaxii instantizována spolu s generátory stimulů a hvězdami zobrazujícími výstup, které jsou dostupné v knihovně systému Ptolemy.

Posledním úkolem, který je nutné provést před vlastním provedením simulace, je výběr implementace pro každý CFSM. V případě funkční simulace stačí přiřadit každému CFSM jednotkové zpoždění. To se provede tak, že se pro každý CFSM zvolí implementace v hardwaru. Simulační mechanismus také vyžaduje, aby byl přidělen procesor pro softwarové rozdělení.

V tomto okamžiku již lze přikročit ke spuštění vlastní simulace. K tomu je třeba zadat příkaz **:run** a nastavit čas ukončení simulace (počet provedených hodinových cyklů). Tato fáze kosimulace se nazývá **funkční ladění** (functional debugging). Jejím účelem je ověřit, zda chování návrhu odpovídá neformální specifikaci.

Po provedení fáze funkčního ladění je zapotřebí definovat architekturu návrhu, tj. hardware/software dělení a výběr procesoru pro softwarovou část. Architektura systému může být definována interaktivně v rámci systému Ptolemy. Je nutné nastavit čtyři parametry v testovací galaxii. Prvním z nich je parametr Clock_freq, který udává frekvenci hodin v MHz pro procesor, který bude použit hvězdami implementovanými v softwaru. Druhým parametrem je parametr SCHEDULER. Ten definuje typ plánování v reálném čase (real-time scheduling policy), který bude použit pro hvězdy implementované v softwaru. Třetí parametr, Firingfile, definuje adresář, do kterého bude umístěn soubor, který obsahuje počáteční a koncový čas každého provedeného přechodu každého softwarového CFSM. Poslední parametr, Overflowfile, definuje adresář, do kterého bude umístěn soubor, který obsahuje informace o ztracených událostech. V tomto okamžiku je již možné spustit i časovou simulaci.

4.3 Překlad netlistu systému Ptolemy do formátu SHIFT

Jakmile jsou výsledky simulace daného hardware/software rozdělení uspokojivé, je třeba netlist systému Ptolemy přeložit do formátu SHIFT, který bude použit následnými kroky syntézy.

5 Formální verifikace

Program POLIS je schopen převést asynchronní síť modelů CFSM na ekvivalentní synchronní síť konečných stavových strojů (FSM). Tato reprezentace je zapsána v souboru ve formátu BLIF-MV a může být následně předána verifikačnímu nástroji VIS (Verification Interacting with Synthesis).

6 Syntéza softwaru a hardwaru v prostředí POLIS

6.1 Syntéza softwaru

Syntéza softwaru v prostředí POLIS je založena na grafech toku řízení a dat (control-data flow graph), které se nazývají S-grafy. Účelem S-grafu je specifikovat přechodovou funkci jednoho CFSM. S-graf je orientovaný acyklický graf, který se skládá ze čtyř typů uzlů. První dva typy jsou uzel BEGIN, který odpovídá počátečnímu místu grafu (source), a uzel END, který odpovídá koncovému místu graf (sink). Třetí typem uzlu je TEST, který odpovídá funkci, která je definována nad množinami vstupních a výstupních proměnných S-grafu. Z uzlů tohoto typu vychází tolik hran, kolika hodnot může příslušná funkce nabývat. Posledním typem uzlu je ASSIGN. Tomu je přiřazena jedna proměnná z množiny výstupních proměnných a funkce, jejíž hodnota je přiřazena této výstupní proměnné. Každý uzel typu ASSIGN má právě jednu výstupní hranu. Průchodem S-grafu z uzlu BEGIN do uzlu END lze vypočítat funkci, kterou daný S-graf reprezentuje. Před začátkem průchodu S-grafem jsou všechny výstupní proměnné nastaveny na nedefinovanou hodnotu.

Odhadovací nástroje poskytují přesné odhady velikosti a doby provedení (execution time) výsledného kódu v případě, že každý uzel S-grafu odpovídá základnímu bloku kódu. Navíc by mělo platit, že interakce mezi jednotlivými základními bloky je omezená.

Po provedení optimalizace je S-graf přeložen do přenositelného kódu v jazyce C. Poté je potřeba použít kompilátor pro implementaci a optimalizaci tohoto kódu do specifické instrukční sady, závislé na nějakém určitém mikrokontroléru.

6.2 Real-time operační systém

Spolupráce mezi množinou souběžných úloh na jednoprocesorovém systému vyžaduje nějaký mechanismus plánování. V případě vestavěných systémů musí toto plánování většinou splňovat určitá časová omezení (timing constraints). Techniky plánování pro real-time systémy lze rozdělit na statické a dynamické.

V případě statického plánování jsou úlohy prováděny v pevném, předem stanoveném pořadí. V případě dynamického plánování je pořadí provádění úloh určeno v době běhu programu. Je také možné použít pro různé úlohy různé priority, kdy je úloha pro následující provádění vybrána z množiny připravených (ready) úloh, podle její priority. Priorita úlohy představuje míru naléhavosti úlohy. Může být určena staticky, tj. v době překladu, nebo dynamicky, tj. v době běhu programu.

Dynamické plánování může rozdělit do dvou skupin. Pokud aktuálně běžící úloha může být odložena (suspended) kvůli tomu, že se do stavu připravenosti (readiness) dostane jiná úloha s vyšší prioritou, potom jde o preemptivní (pre-emptive) dynamické plánování. V opačném případě mluvíme o nepreemptivním (non-pre-emptive) dynamickém plánování.

Statické plánování má výhodu v tom, že nespotřebovává žádnou práci procesoru pro výpočet plánu v době běhu (čas potřebný pro výpočet plánu v době před prováděním programu nemusí být sice zanedbatelný, ale je vyžadován pouze jednou). Je však nutné znát předem čas připravenosti jednotlivých úloh. Tato podmínka bývá splněna v případě systémů orientovaných na tok dat (data-flow-oriented systems). Zde se úlohy stávají připravenými na základě příchodu vstupních událostí, které obecně přicházejí v pravidelných proudech (streams). Pokud doba připravenosti úloh není předvídatelná, je statické plánování velmi neefektivní, protože mnoho procesorového času je věnováno pouze pollingu událostí. V takovém případě je výhodnější dynamické plánování.

Syntéza real-time operačního systému (RTOS), která se v programu POLIS provede po zadání příkazu **gen_os**, je složena ze tří kroků. Nejprve jsou přiřazeny vstupní/výstupní (I/O) porty signálům, které byly definovány jako rozhraní mezi softwarovými CFSM a okolím systému. Dále se přiřadí jednotlivým CFSM priorita a odhadnuté maximální doby provedení. Nakonec se vygeneruje program v jazyce C, který implementuje ovladače I/O a náležitě plánuje jednotlivé CFSM.

6.3 Syntéza hardwaru

Syntéza hardwaru vytváří Mooreův stroj pro každý CFSM. Následně může být použit proces logické syntézy za účelem provedení optimalizace. Program POLIS podporuje rychlé prototypování tak, že je schopen produkovat výstupní netlist ve formátu XNF, který je možné optimalizovat pro danou architekturu obvodů FPGA. Tím je umožněna implementace hardwarových CFSM v obvodech FPGA firmy Xilinx. Dále jsou podporovány také výstupní netlisty ve formátech BLIF, Verilog a VHDL.

Pro zápis netlistu, nacházejícího se aktuálně v paměti, do souboru ve formátu XNF slouží příkaz **write_xnf**.

6.4 Metodologie rychlého prototypování založená na APTIX FPIC

Návrhové prostředí POLIS nabízí podporu pro rychlé prototypování vestavěných systémů s využitím programovatelné propojovací desky APTIX, emulace mikrokontrolérů v obvodu (in-circuit emulation) a čipů FPGA.

Pro softwarovou část systému je syntetizovaný kód v jazyce C přeložen, načten do emulátoru a laděn. Pro hardwarovou část systému je syntetizovaná a optimalizovaná logika namapována do několika intermediárních formátů, které jsou následně převedeny do formátu XNF. Pro každý čip FPGA je vygenerován jeden soubor ve formátu XNF. V této době ještě nejsou dostupné časové informace. Následuje mapování formátu LCA do příslušného formátu pro X-checker programování cílových čipů FPGA. Ve stejné době je zpětně okomentován soubor ve formátu Ca, který již obsahuje časové informace, které jsou závislé na cílové technologii, a balíčkové informace (package information). Nyní je možné provést časovou simulaci fyzických modelů čipů FPGA firmy Xilinx.

Program POLIS kromě FPGA čipů firmy Xilinx podporuje také čipy FPGA firmy ACTEL. Pro

zápis netlistu do souboru ve formátu pro tyto čipy FPGA slouží příkaz **act_map**.

Propojení různých částí systému (mikrokontoléru a jednotlivých čipů FPGA) je možno získat pomocí komerčního editoru schémat. Získané schéma je použito pro vytvoření netlistu systému pro programování zařízení FPIC desky APTIX. Fyzický prototyp je v této chvíli hotový.

7 Návrh aritmeticko-logické jednotky

V jazyce ESTEREL jsem zapsal několik modulů, které provádí základní logické operace. Jde o modul m_and, který představuje dvouvstupové hradlo AND, modul m_or, reprezentující dvouvstupové hradlo OR a modul m_xor, který implementuje funkci dvouvstupového hradla XOR. Dále jsem vytvořil návrh dvoubitové sčítačky (modul m_add) a dvoubitové násobičky (modul m_mult).

Všechny výše zmíněné moduly jsou uloženy v samostatných souborech. Splnění této podmínky je vyžadováno programem POLIS. Obecně však jazyk ESTEREL umožňuje definovat v jednom souboru více modulů.

Dále jsem vytvořil modul ealu, který odpovídá aritmeticko-logické jednotce (ALU). Modul ealu má celkem 7 vstupních pinů, z nichž 2 jsou určeny pro první operand A, další 2 pro druhý operand B a zbývající 3 pro určení operace, která se má provést. Tato ALU má dále 2 výstupní piny pro výsledek operace C a 1 pin, který indikuje zda došlo při operaci sčítaní k přetečení (výstup flag_C).

V modulu ealu jsou instantizovány moduly m_and, m_or, m_xor, m_add a m_mult, které provádějí výše popsané operace. Všechny tyto operace jsou definovány behaviorálně s využitím logických funkcí and, or a xor. Příkaz pause před emitováním výstupního signálu je používán, aby se zabránilo vzniku smyčky s nulovým zpožděním (instantaneous loop). Těla všech modulů jsou totiž umístěna ve smyčce loop <tělo smyčky> end loop. To znamená, že pokud provádění těla smyčky skončí, začne okamžitě nové provádění této smyčky od jejího začátku. Je tedy třeba, aby se v těle smyčky vyskytoval alespoň jeden příkaz, který má nenulové zpoždění.

V modulu ealu se nachází také prvek, který na základě hodnot vstupů určujících typ operace, která se má provést, přiřadí výstupy modulu ealu signálům, které jsou připojeny na vstup jednotky, která je zvolena, aby provedla operaci. Všechny vstupy, výstupy i vnitřní signály jsou typu boolean.

8 Číslicový FIR filtr

FIR filtr je takový typ filtru, který má konečnou odezvu na jednotkový impuls (Kroneckerova delta funkce). Kroneckerova delta funkce je funkce delta(k), která má pro k = 0 hodnotu 1 a pro všechny ostatní hodnoty k má hodnotu 0.

Diferenční rovnice FIR filtru má tvar

y[n] = b0 * x[n] + b1 * x[n-1] + ... + bN-1 * x[n - (N - 1)] + bN * x[n - N],

kde x[n] představuje vstupní signál, y[n] je výstupní signál a bi jsou koeficienty filtru.

Konstanta N označuje řád filtru. FIR filtr, který je řádu N, musí mít (N + 1) koeficientů, a tedy musí uchovávat historii (N + 1) minulých hodnot vstupního signálu.

8.1 Počáteční návrh FIR filtru

Pro výpočet koeficientů navrhovaného FIR filtru jsem použil program FIRDsgn, jehož autorem je Jialong He [1].

Jako typ FIR filtru jsem zvolil dolní propust s horní mezní frekvencí (cutoff freqeuncy) 500 Hz. Vzorkovací frekvenci jsem nastavil na hodnotu 40 000 Hz, neboť jsem vycházel z předpokladu, že navrhovaný FIR filtr bude použit pro filtraci zvukového signálu, přičemž horní hranice slyšitelnosti zvuku lidským sluchem je 20 000 Hz. Minimální vzorkovací frekvence pro tento rozsah je tedy 40 000 Hz (vzorkovací toerém). Řád filtru jsem nastavil na hodnotu 500.

8.2 Implementace FIR filtru

Zápis zdrojového programu FIR filtru v jazyce Esterel je v souboru fir.strl. Celý FIR filtr je tvořen jedním modulem (module fir). Tento modul má jeden vstup a jeden výstup. Na vstupu X jsou očekávány vzorky diskrétního sinusového signálu. Na výstupu Y jsou postupně emitovány hodnoty přefiltrovaného signálu. Hodnoty vstupu X i výstupu Y jsou definovány jako typ integer, jsou však následně pomocí pomocného souboru předefinovány na typ float. Jazyk Esterel sice umožňuje přímo použít typ float ve zdrojovém programu, ale toto není podporováno v programu Polis. U všech vstupů, výstupů a proměnných, které mají být typu float, je třeba nejprve ve zdrojovém programu v Esterelu deklarovat jejich typ jako integer a následně je možné použít v pomocném souboru předefinování jejich typu na jiný typ, v tomto případě na typ float. Přetypování vstupu X a výstupu Y z typu integer na typ float je provedeno v souboru fir.aux (pomocný soubor je určen makrem INIT_AUX v souboru Makefile.src) těmito řádky:

nb X 31 float in fir
nb Y 31 float in fir

Tyto příkazy určují, že vstupní signál X a výstupní signál Y budou typu float a že tento typ bude mít 32 bitů (1 znaménkový bit + 31 bitů). V případě, že by před číslem 31 bylo uvedeno klíčové slovo "unsigned", potom by hodnota daného signálu měla 31 bitů. Klauzule "in fir" informuje Polis, že toto přetypování se týká proměnných s těmito názvy pouze v modulu fir a nikoliv proměnných se stejnými názvy v jiných modulech v rámci návrhu.

Řád FIR filtru jsem zvolil jako hodnotu 500, počet koeficientů filtru je tedy 501. Koeficienty jsou zapsány v uživatelské (user-defined) funkci GET_COEF, která je v souboru fir.strl pouze deklarovaná. Její tělo je uvedeno v souboru loc_types.c (jedná se o soubor, který je použit pro definici uživatelských funkcí a je určen makrem POLISLIBS v souboru Makefile.src). Funkce GET_COEF ma jediný parametr "index" typu integer, na základě kterého bude touto funkcí vrácen některý z koeficientů filtru. Koeficienty filtru jsou ukládány do struktury "array_coef" typu Tarray501. Tento typ je deklarovaný v souboru loc_types.h (soubor loc_types.h musí obsahovat deklarace všech uživatelských typů a všech uživatelských funkcí, které jsou použity v programu zapsaném v jazyce Esterel):

```
typedef struct
{
  float array[POCET_COEF];
} Tarray501;
```

Konstanta POCET_COEF je také definovaná v souboru loc_types.h. Její hodnota je 501. Struktura typu Tarray501 tedy obsahuje pole 501 hodnot typu float.

Proměnná, do které se ukládá minulých 501 hodnot ze vstupu X, je také typu Tarray501 a je deklarovaná přímo v zápisu modulu fir v souboru fir.strl. Jmenuje se "pole_X". Pro přístup k prvkům pole obsaženého v této proměnné slouží dvě další uživatelské funkce: GET_POLE_X a PUT_POLE_X.

Funkce GET_POLE_X má dva parametry. Prvním z nich je celočíselný parametr "index", který udává pozici požadované minulé vstupní hodnoty v poli array proměnné pole_X. Druhý parametr je typu Tarray501, který umožňuje zvolit proměnou typu Tarray501, nad kterou bude tato funkce provedena. Vzhledem ke jménu a účelu této funkce musí být tomuto parametru vždy přiřazena proměnná "pole_X". Smyslem funkce GET_POLE_X je vrátit z minulých vstupních hodnot FIR filtru tu, která je specifikována hodnotou parametru index. Protože pole array ve struktuře Tarray501 má právě 501 položek, může parametr "index" nabývat pouze hodnot v rozsahu od 0 do 500.

Funkce PUT_POLE_X má opačný význam než funkce GET_POLE_X - má za úkol vkládat aktuální hodnotu vstupu X do proměnné pole_X. Toto vkládání se ale neprovádí na předem zvolenou pozici v poli array proměnné pole_X, nýbrž vždy na první pozici tohoto pole, tj. vždy do položky na indexu 0. Z tohoto důvodu, na rozdíl od funkce GET_POLE_X, nepotřebuje funkce PUT_POLE_X parametr "index". Přesto má tato funkce rovněž dva parametry. Prvním z nich je parametr "hodnota", který je v souboru fir.aux přetypován na typ float. Nabývá hodnoty, kterou má vstup X v okamžiku volání funkce PUT_POLE_X (to je zaručeno tím, že se do tohoto parametru předává výraz ?X, který reprezentuje aktuální hodnotu signálu X). Druhý parametr funkce PUT_POLE_X se jmenuje "array_X" a je typu Tarray501. Význam tohoto parametru je totožný jako u funkce GET_POLE_X a také do něj lze přiřadit výhradně proměnnou pole_X.

Funkce PUT_POLE_X zachází s proměnnou "pole_X" jako s cirkulárním bufferem. Jestliže je zapotřebí uchovávat v každém okamžiku minulých 501 hodnot vstupního singálu X, potom po následujícím zavolání funkce PUT_POLE_X již není nutné uchovávat v paměti hodnotu vstupu X, která se vyskytovala v poli array v proměnné "pole_X" na indexu 500 v době před výše zmíněným voláním funkce PUT_POLE_X.

Tělo funkce PUT_POLE_X se skládá ze dvou částí, z nichž první je tvořena cyklem for, který

každému prvku pole array proměnné "pole_X", s výjimkou prvku s indexem 0, přiřadí hodnotu prvku s indexem o 1 menším. Druhou část těla této funkce tvoří přiřazovací příkaz, který přiřadí prvku s indexem 0 hodnotu, která je předávána prostřednictvím paramteru "hodnota". Funkce PUT_POLE_X vrací jako návratovou hodnotu celou modifikovanou strukturu typu Tarray501, která je v zápisu modulu fir přiřazena do proměnné "pole_X".

Celý zápis modulu fir je, s výjimkou deklarací na jeho začátku, tvořen konstruktem "every X do <p> end every". Tento příkaz čeká na výskyt události na vstupu X a potom provede tělo <p>. Znamená to, že se vše, co se nachází mezi "every X do" a "end every" provede znovu při každém dalším výskytu události na vstupu X. Tělo <p> v případě modulu fir provádí výpočet hodnoty výstupního signálu Y jako odezvu na nově příchozí vstupní vzorek.

Prvním krokem, který se po výskytu nové události na vstupu X provede, je aktualizace proměnné "pole_X" pomocí funkce PUT_POLE_X.

Aktuální hodnota, která má být emitována jako hodnota výstupního signálu Y, se ukládá do proměnné "odezva" (v Esterelu deklarovaná jako integer, v souboru fir.aux přetypovaná na typ float). Proměnná "odezva" je před začátkem každého provádění výpočtu aktuálního výstupu FIR filtru inicializována na nulovou hodnotu.

Výpočet aktuální hodnoty výstupního signálu Y je prováděn cyklem repeat. Počet jeho iterací je určen hodnotou celočíselné konstanty POCET_KOEFICIENTU, jejíž hodnota je v tomto případě 501.

Tělo tohoto příkazu repeat využívá 3 pomocné proměnné - "j", "k" a "pom". Celočíselná proměnná pom je deklarována před tímto cyklem repeat a inicializována na nulovou hodnotu. Je použita pro identifikaci iterace tohoto cyklu repeat. Po provedení každé iterace je její hodnota inkrementována o 1. Nabývá tedy hodnot z rozsahu od 0 do 500.

Při každé iteraci cyklu repeat je do pomocné proměnné "j" pomocí funkce GET_COEF vložena hodnota koeficientu, který je určen hodnotou pomocné proměnné "pom" (proměnná "pom" má zde význam hodnoty indexu pole koeficientů v proměnné array_coeff) a do proměnné "k" je pomocí funkce GET_POLE_X vložena jedna z minulých vstupních hodnot FIR filtru, a to rovněž na základě hodnoty proměnné "pom" (zde má proměnná "pom" význam hodnoty indexu požadovaného prvku pole minulých hodnot vstupního signálu X v proměnné "pole_X").

Hodnota proměnné "odezva" je ihned poté inkrementována o hodnotu součinu proměnných "j" a "k" a proměnná "pom" je inkrementována o 1.

Posledním příkazem, který je nutné provést v každé iteraci cyklu repeat, je příkaz "pause". Ten je vyžadován z toho důvodu, aby cyklus repeat neměl nulové zpoždění. Jazyk Esterel nedovoluje použít smyčku, která skončí ve stejném okamžiku, ve kterém začala (instantaneous loop). Aby měla smyčka nenulové trvání, musí se v ní vyskytovat alespoň jeden příkaz, který má nenulové trvání. Jedním z těchto příkazů je příkaz "pause", který pozastaví provádění programu až do příštího výskytu signálu tick. Signál tick je deklarován implicitně. Reprezentuje aktivaci hodin reaktivního programu. Je v každém okamžiku aktivní.

Po tomto kroku je buď spuštěna další iterace cyklu repeat, nebo v případě, že má proměnná "pom" hodnotu (POCET_KOEFICIENTU - 1), celý cyklus repeat končí.

Výsledná hodnota proměnné "odezva" je poté emitována jako hodnota výstupního signálu Y.

8.3 Simulace FIR filtru v simulačním prostředí Ptolemy

Simulace obsažené v této části jsou pouze funkčními simulacemi (functional simulation), které mají za cíl ověřit správnou funkci navrženého FIR filtru.

Příkazem "makemakefile" v příkazovém řadku operačního systému se s využitím hodnot maker v souboru Makefile.src vygeneruje soubor makefile (soubor Makefile.src je specifický pro každý projekt a musí být vytvořen autorem projektu).

Zadáním příkazu "make ptl" ve stejném adresáři, ve kterém byl zadán příkaz "makemakefile", se vygenerují simulační soubory pro systém Ptolemy, a to ze zdrojových souborů s moduly návrhu zapsanými v jazyce Esterel, jakož i pomocných souborů uvedených v příslušných makrech v souboru Makefile.src. Tyto simulační soubory jsou vytvořeny v adresáři, který se jmenuje ptolemy a který je přímým podadresářem hlavního adresáře projektu. Zároveň se v podadresáři ptolemy vygeneruje také návrhová plocha (facet) pro projekt, jejíž jméno je identické se jménem projektu, které je uvedeno v makru TOP v souboru Makefile.src, a facet pro testování projektu, jenž se pojmenuje jménem projektu, před které se připojí "test_".

Příkaz "make ptl" nejprve pomocí příkazu "strl2shift" přeloží zdrojové soubory s moduly zapsanými v jazyce Esterel do formátu SHIFT. Poté se spustí interpret příkazů Polisu, kterému je předána sekvence několika příkazů. Nejprve se načte soubor ve formátu SHIFT pro daný modul příkazem "read_shift <jméno modulu>". Pro simulaci se používá vygenerovaný kód v jazyce C, a proto se v tomto okamžiku přiřadí všem CFSM softwarová implementace pomocí příkazu "set_impl -s". Další příkaz "partition" má za úkol vybudovat vnitřní datové struktury pro syntézu hardwaru a softwaru (v tomto případě pouze pro software). Následujícím příkazem je "build_sg", který vytvoří graf toku řízení/dat (control/data flow graph). Potom přichází na řadu příkaz "sg_to_c", který vytvoří soubory v jazyce C. Aby byly do simulace zahrnuty informace o hodinových cyklech charakteristické pro nějaký zvolený procesor, je potřeba zavolat tento příkaz s volbou "-D".

Po nainstalování programu Ptolemy ze zdrojových souborů se v souboru Makefile.strl.beg, ze kterého se po zadání příkazu "makemakefile" generuje makefile, tata volba nevyskytovala. Jelikož jsem chtěl informace o hodinových cyklech zahrnout do simulace, přidal jsem do zmíněného souboru za příkaz "sg_to_c" volbu "-D" a před tento příkaz jsem zařadil ještě navíc další dva příkazy.

Prvním z těchto dvou příkazů je příkaz "read_cost_param", který načte množinu procesorově-specifických parametrů ze souboru, který charakterizuje daný procesor. Tyto soubory jsou obsaženy v adresáři $POLIS/polis_lib/estm, kde $POLIS označuje kořenový adresář instalace programu Polis. Na výběr jsou zde čtyři soubory charakterizující nějaký procesor. Těmito procesory jsou Motorola 68332, Motorola 68hc11, MIPS R3000 a SPARC.

Druhým příkazem je "print_cost", který provede odhad minimálního počtu cyklů hodin, maximálního počtu cyklů hodin a velikosti kódu pro každý S-graf.

Pro výběr procesoru z uvedených čtyř možností jsem vložil do souboru Makefile.strl.beg ještě další dva příkazy, a to příkaz "set arch" a příkaz "set archsuffix".

Příkaz "set" slouží k nastavení proměnných příkazového interpretu Polisu na určitou hodnotu. Různé příkazy používají proměnné prostředí Polisu pro různé účely. Proměnná "arch" specifikuje instrukční sadu cílového mikroprocesoru. Odpovídající soubor z adresáře $POLIS/polis_lib/estm bude použit k vytvoření operačního systému. Proměnná "archsuffix" specifikuje jméno architektury

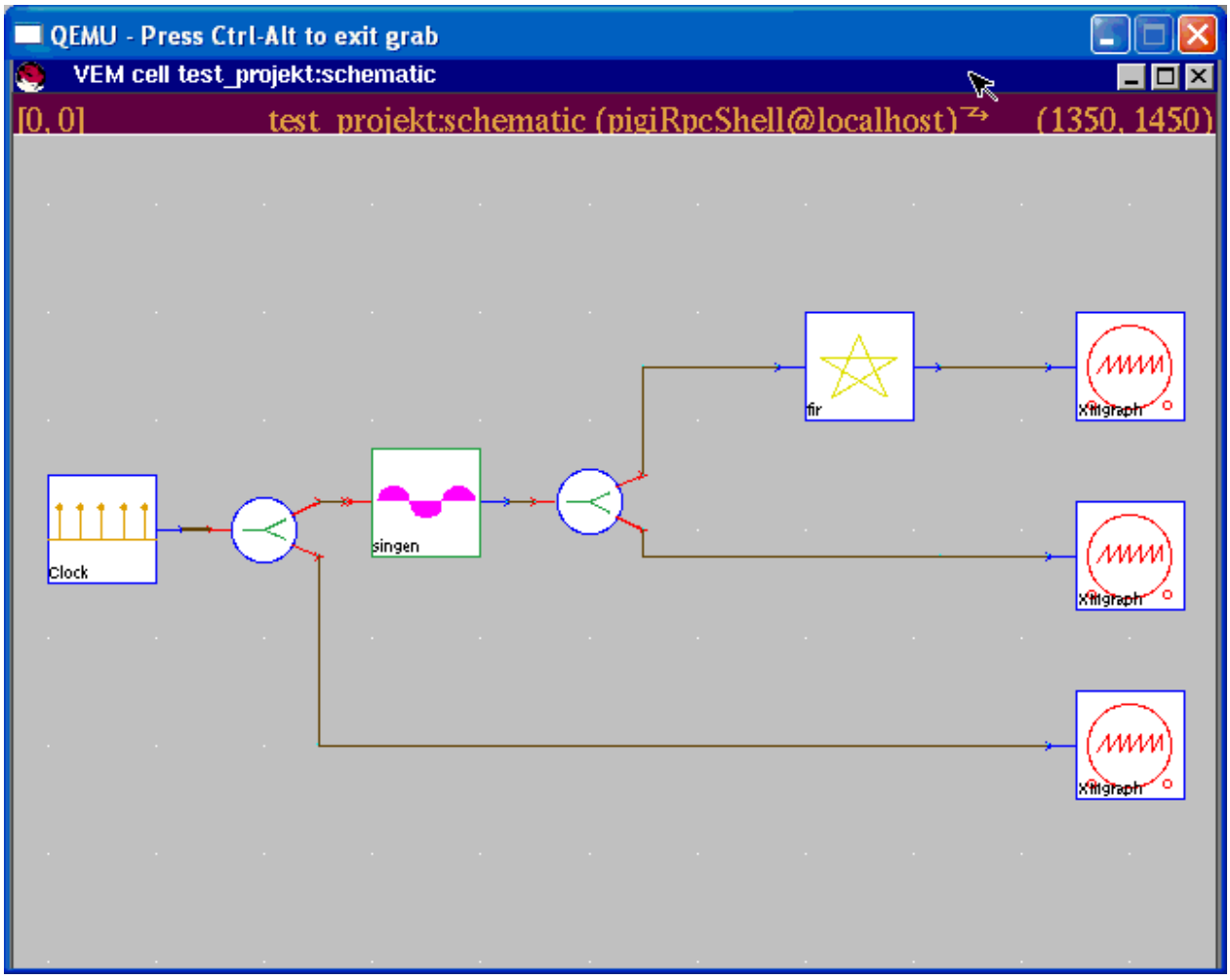
cílového mikroprocesoru na implementační úrovni. Odpovídající soubor obsahuje informace o výstupních pinech a o velikosti paměti.

Po přepnutí do podadresáře ptolemy se příkazem pigi se jménem facetu projektu spustí grafické prostředí systému Ptolemy a otevře se zároveň facet tohoto projektu.

Pro každý CFSM je třeba vytvořit ikonu. K tomu slouží příkaz :make-star, který lze spustit tak, že se kurzor myši umístí do nějakého facetu a stiskne se na klávesnici znak hvězdička. Na základě příkazu :make-star se objeví na obrazovce formulář. V něm je nutné nastavit doménu na hodnotu DE (pro všechny projekty vygenerované programem Polis), přičemž jméno hvězdy musí být stejné jako jméno CFSM (tedy jako jméno modulu v Esterelu). Paletou, do které se má vytvořená ikona umístit, by měla být user.pal.

Paleta user.pal je speciálním případem galaxie, která obsahuje instanci ikony každé hvězdy a každé galaxie v rámci aktuálního vesmíru.

Do facetu test_projekt jsem umístil ikonu hvězdy fir a provedl připojení dalších komponent, které jsou standardní součástí systému Ptolemy. Výsledné schéma je zachyceno na následujícím obrázku.



Obr. 8.1 Testovací facet pro hvězdu fir

Zdrojem hodinových impulsů je hvězda "Clock", která umožňuje nastavit interval mezi jednotlivými impulsy pomocí parametru "interval" a amplitudu impulsů pomocí parametru "magnitude". V tomto projektu mají oba tyto parametry ponechané výchozí hodnoty: interval = 1.0 a magnitude = 1.0.

Na výstup hvězdy "Clock" je připojena hvězda "Fork.output=2". Její funkce spočívá pouze v tom, že vše, co přichází na její vstup, posílá nezměněno na oba své výstupy. Účelem dolní větve je propustit dále úvodní hodinový signál. Signál vycházející z horního výstupu hvězdy "Fork.output=2" je připojen na vstup galaxie "singen". Signál vycházející z jejího dolního výstupu vstupuje do hvězdy "XMgraph.input=1", jejíž funkcí je zobrazit příchozí vstupní signál do grafu, který se vykreslí v novém okně.

Galaxie "singen" je tvořena jinou galaxií "singen" domény SDF, která je umístěna v červí díře (wormhole) domény DE. Galaxie "singen" domény SDF je tvořena hvězdou "Ramp" domény SDF, která na svém výstupu generuje "rampový" signál, jehož počáteční hodnota je určena hodnotou parametru value (výchozí hodnota je 0.0). Při každém dalším provedení (firing) této hvězdy je hodnota výstupního signálu inkrementována o hodnotu dalšího parametru "step" (výchozí hodnota je 1.0). Výstupní signál hvězdy "Ramp" je připojen na vstup hvězdy "Sin" domény SDF.

Hvězda "Sin" domény SDF vypočte hodnotu funkce sinus, jíž je jako argument předána hodnota vstupního signálu, která se považuje za velikost úhlu v radiánech.

Galaxie "singen" domény SDF má tři parametry. Prvním z nich je parametr "sample_rate", který udává úhlovou vzorkovací frekvenci (vzorkovací frekvence by měla být vždy větší než dvojnásobek nejvyšší frekvence obsažené ve vzorkovaném signálu, aby bylo možné správně zrekonstruovat původní spojiý signál z navzorkovaného signálu pomocí rekonstrukčního filtru (dle vzorkovacího teorému)). Druhým parametrem je "frequency", jehož hodnota představuje úhlovou frekvenci generovaného sinusového signálu vzhledem k úhlové vzorkovací frekvenci dané parametrem "sample_rate". Třetím parametrem je "phase_in_radians", který uchovává velikost počáteční fáze generované sinusoidy (výchozí hodnota je 0.0). Tyto tři parametry jsou pomocí hierarchického předávání parametrů použity hvězdou "Ramp" uvnitř galaxie "singen" domény SDF. Hvězda "Ramp" má jako svoji počáteční hodnotu "rampového" signálu nastavenu hodnotu parametru "phase_in_radians" galaxie "singen" domény SDF. Parametr "step" hvězdy "Ramp" je vyjádřen jako výraz 2 * PI * frequency / sample_rate, kde frequency a sample_rate jsou hodnoty parametrů "frequency" a "sample_rate" galaxie "singen" domény SDF.

Galaxie "singen" domény SDF tedy generuje sinusový signál podle rovnice

sin(2 * PI * (frequency/sample_rate) * n + phase_in_radians),

kde n je index vzorku signálu a PI označuje Ludolfovo číslo.

Stejný signál, který je generován galaxií "singen" domény SDF, je tedy prostřednictvím červí díry domény DE generován také galaxií "singen" domény DE.

Na výstup galaxie "singen" domény DE je připojena hvězda "Fork.output=2", která duplikuje svůj vstupní signál do dvou stejných signálů na jejím výstupu. Signál z jejího dolního výstupu je připojen na vstup hvězdy "XMgraph.input=1", která zobrazí původní sinusový signál. Signál z jejího horního výstupu je přiveden na vstup hvězdy "fir", jejíž funkce byla popsána výše v tomto textu. Výstupní signál hvězdy "fir" je připojen na vstup třetí instance hvězdy "XMgraph.input=1", která v grafu zobrazí výsledný přefiltrovaný signál.

Parametry galaxie "singen" domény SDF jsem nastavil podle parametrů, které byly vstupem do programu FIRDsgn, jenž byl použit pro výpočet koeficientů FIR filtru. Jelikož vzorkovací frekvence

byla před výpočtem koeficientů nastavena na hodnotu 40 000 Hz a parametr "sample_rate" nepředstavuje vzorkovací frekvenci, nýbrž úhlovou vzorkovací frekvenci (v jednotkách radián za sekundu), a jelikož mezi frekvencí a úhlovou frekvencí platí vztah

úhlová frekvence [rad/s] = 2 * PI [rad] * frekvence [1/s],

vložil jsem do parametru "sample_rate" galaxie "singen" domény SDF hodnotu 80 000 * PI.

Parametr "phase_in_radians" jsem ponechal na výchozí hodnotě 0.0.

Do parametru "frequency" jsem zadával různé hodnoty, abych si na grafickém výstupu simulace ověřil, zda získané výsledky (přefiltrovaný signál) odpovídají předpokládaným výsledkům.

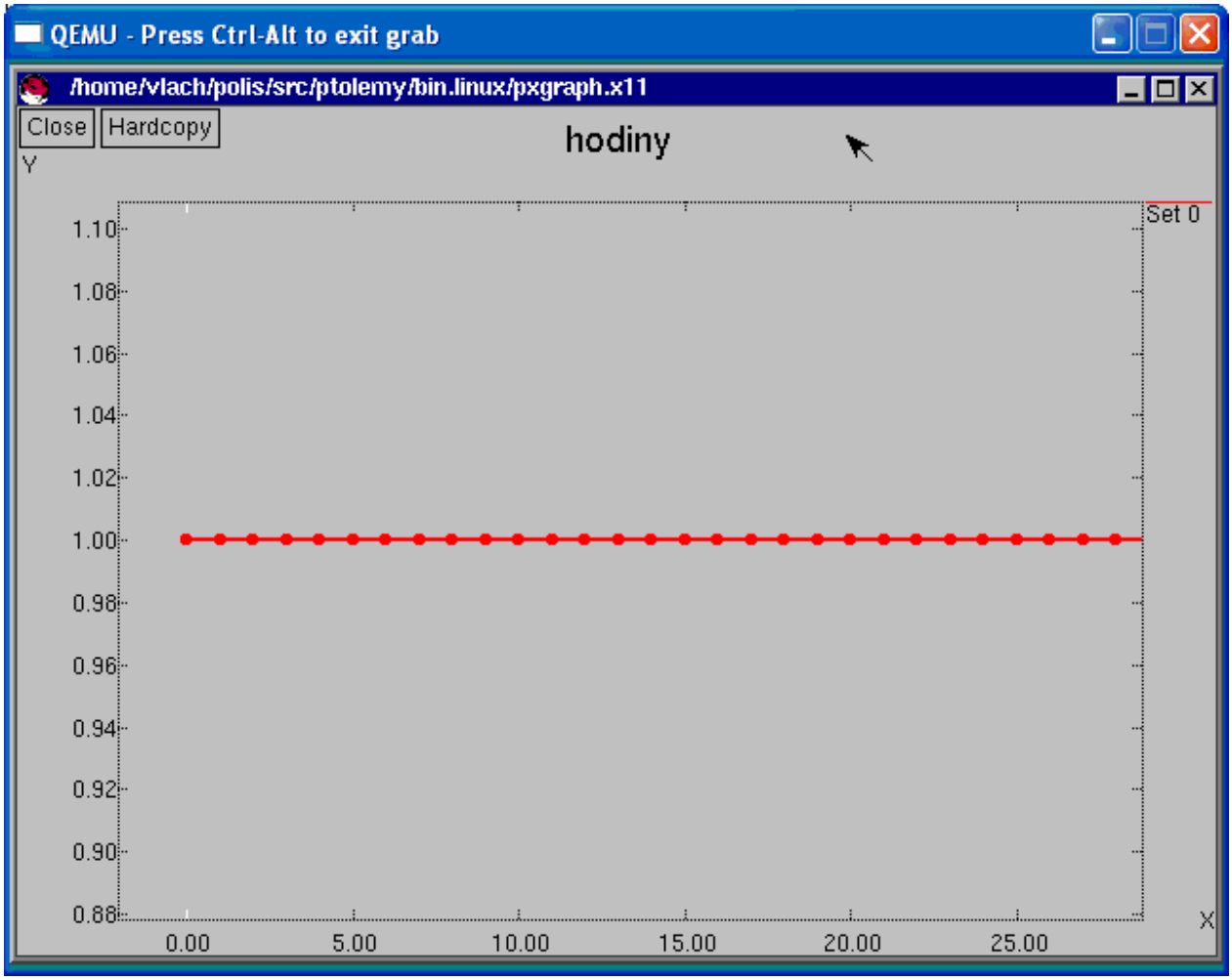
Protože horní mezní frekvence implementovaného FIR filtru je 500 Hz, pokusil jsem se do parametru "frequency" galaxie "singen" domény SDF nejprve vložit takovou hodnotu, aby signál získaný na výstupu FIR filtru byl totožný se signálem vstupujícím do FIR filtru.

Vstupní sinusový signál s frekvencí 100 Hz by tedy měl FIR filtrem projít beze změny. Frekvenci 100 Hz podle výše uvedeného vzorce odpovídá úhlová frekvence 200 * PI [rad/s].

Hodnotu parametru "frequency" galaxie "singen" domény SDF jsem tedy nastavil na 200 * PI. Vlastní simulaci jsem spustil pomocí příkazu :run.

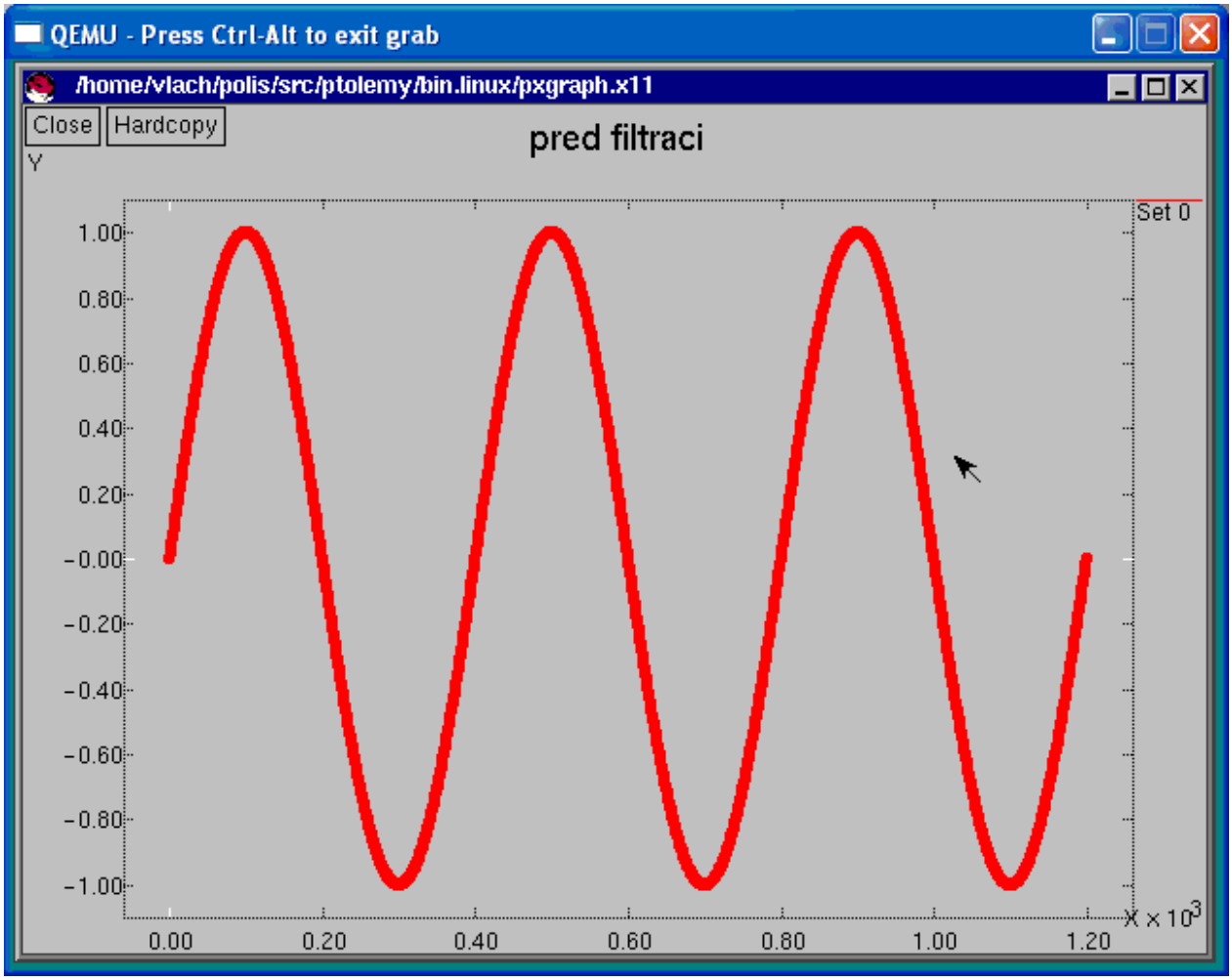
Po skončení simulace se na obrazovce objevila tři nová okna, jedno pro každý výskyt hvězdy "XMgraph.input=1" ve výše uvedeném schématu. V jednom z těchto oken je zobrazen průběh hodinového signálu, ve druhém okně sinusový signál před filtrací a ve třetím okně signál po filtraci FIR filtrem.

Průběh hodinového signálu je pro všechny simulace identický, protože jsem parametry hvězdy "Clock" neměnil. Jde o diskrétní signál, který má vzdálenost mezi sousedními vzorky 1.0, přičemž hodnota všech vzorků je 1.0 (viz obrázek).



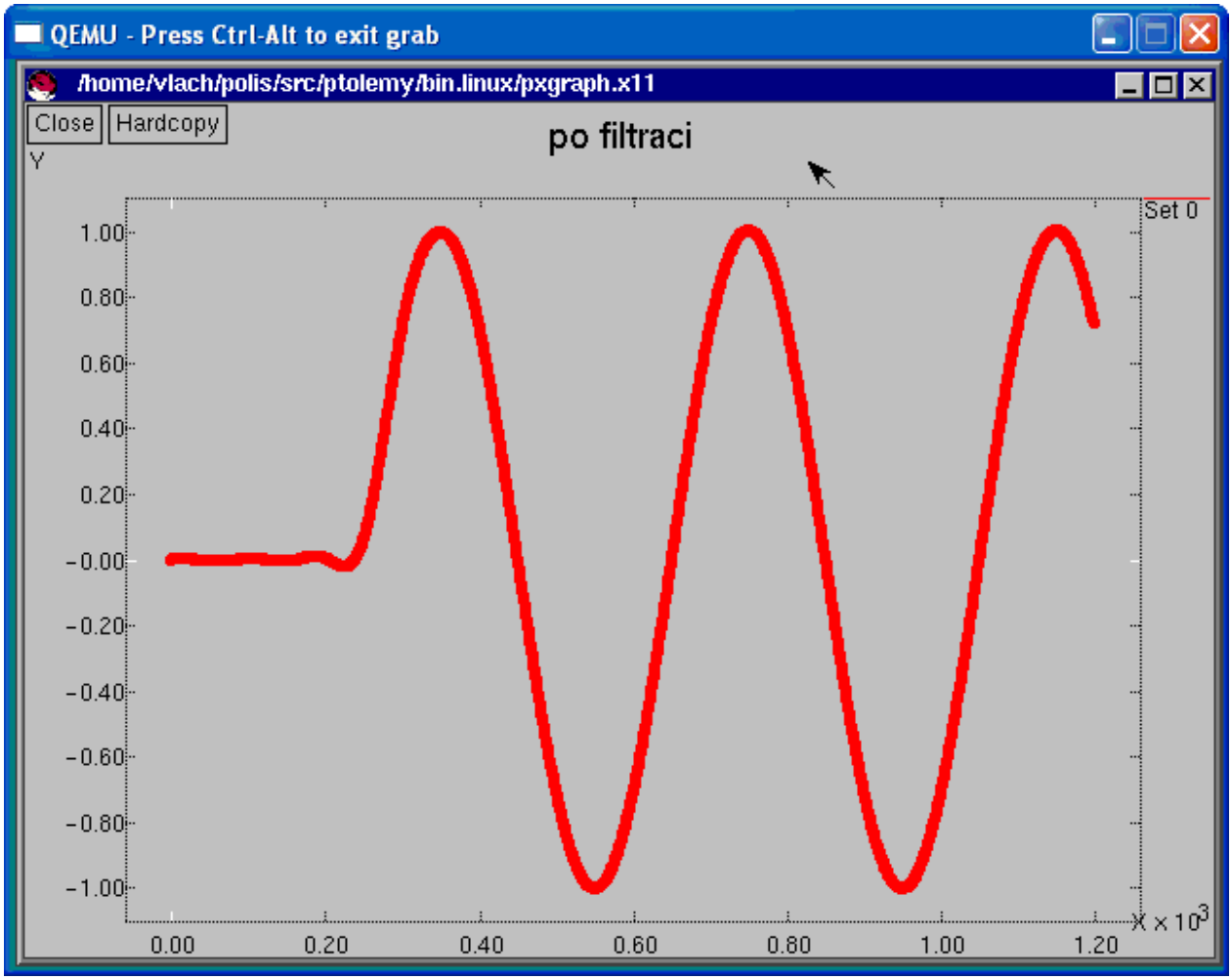
Obr. 8.2 Hodinový signál

Sinusový signál před filtrací odpovídá diskrétní funkci sinus s výše uvedenými parametry. Amplituda tohoto signálu je 1.0 (viz následující obrázek).



Obr. 8.3 Vstupní signál 200 Hz

Signál po filtraci FIR filtrem je na následujícím obrázku.

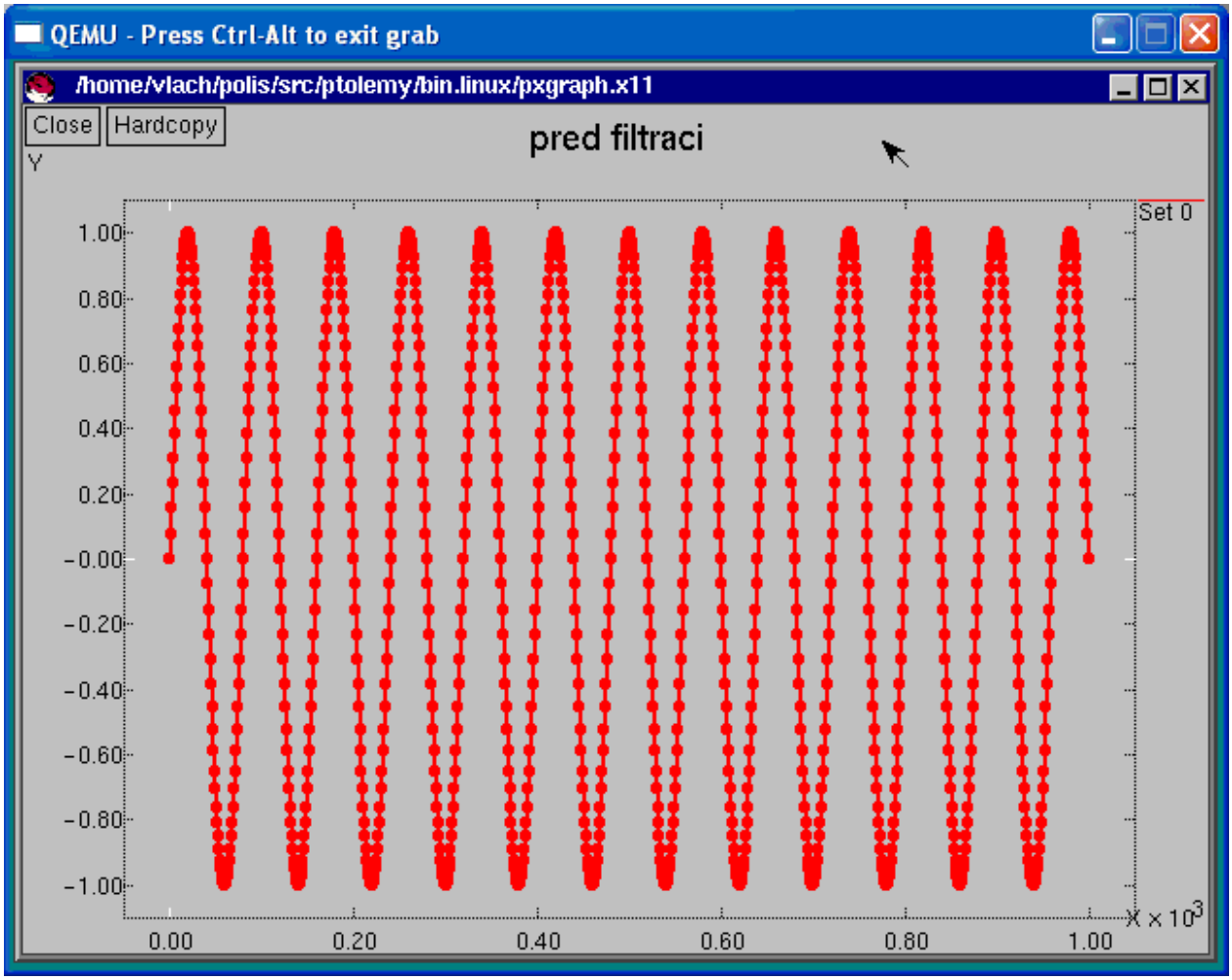


Obr. 8.4 Signál 200 Hz po filtraci

Začátek průběhu přefiltrovaného signálu je důsledkem přechodového děje, jehož příčinou je skutečnost, že prvky pole minulých vstupních hodnot mají na počátku náhodné hodnoty a teprve postupně se toto pole plní příchozími vzorky signálu. Pokud nebude brán v úvahu počáteční přechodový děj, dá se říci, že vstupní signál prošel FIR filtrem na jeho výstup nezměněn, což odpovídá předpokladu, protože frekvence 100 Hz leží uvnitř propustného pásma filtru.

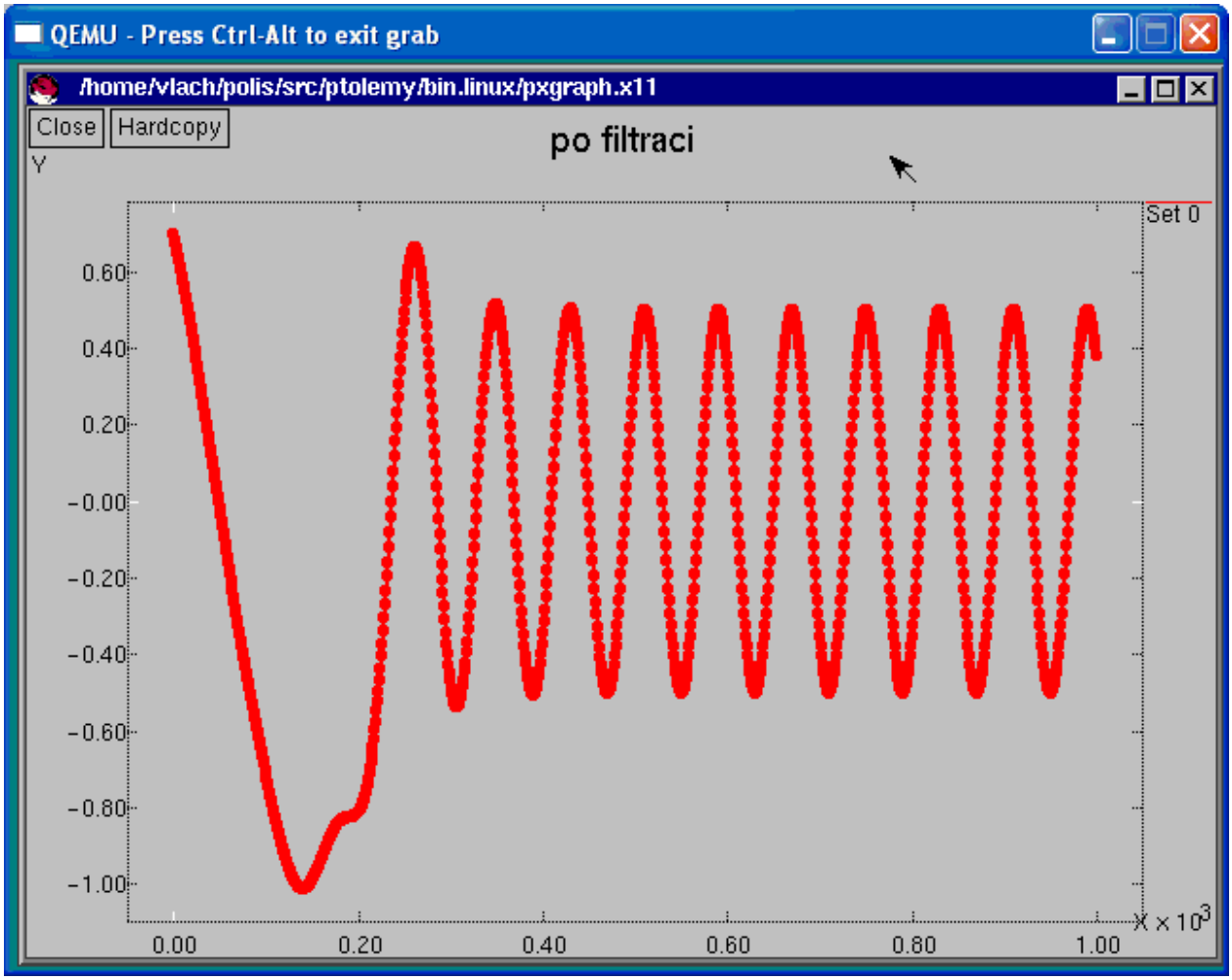
Dále jsem se pokusil nastavit frekvenci vstupního signálu na hodnotu mezní frekvence FIR filtru, což je 500 Hz. Do parametru "frequency" galaxie "singen" domény SDF jsem tedy zadal 1000 * PI.

Vstupní signál vypadá podobně jako v předchozím případě, pouze má kratší periodu (viz následující obrázek).



Obr. 8.5 Vstupní signál 500 Hz

Výstupní signál se však tentokrát od vstupního signálu liší i po počátečním přechodovém ději (viz následující obrázek).

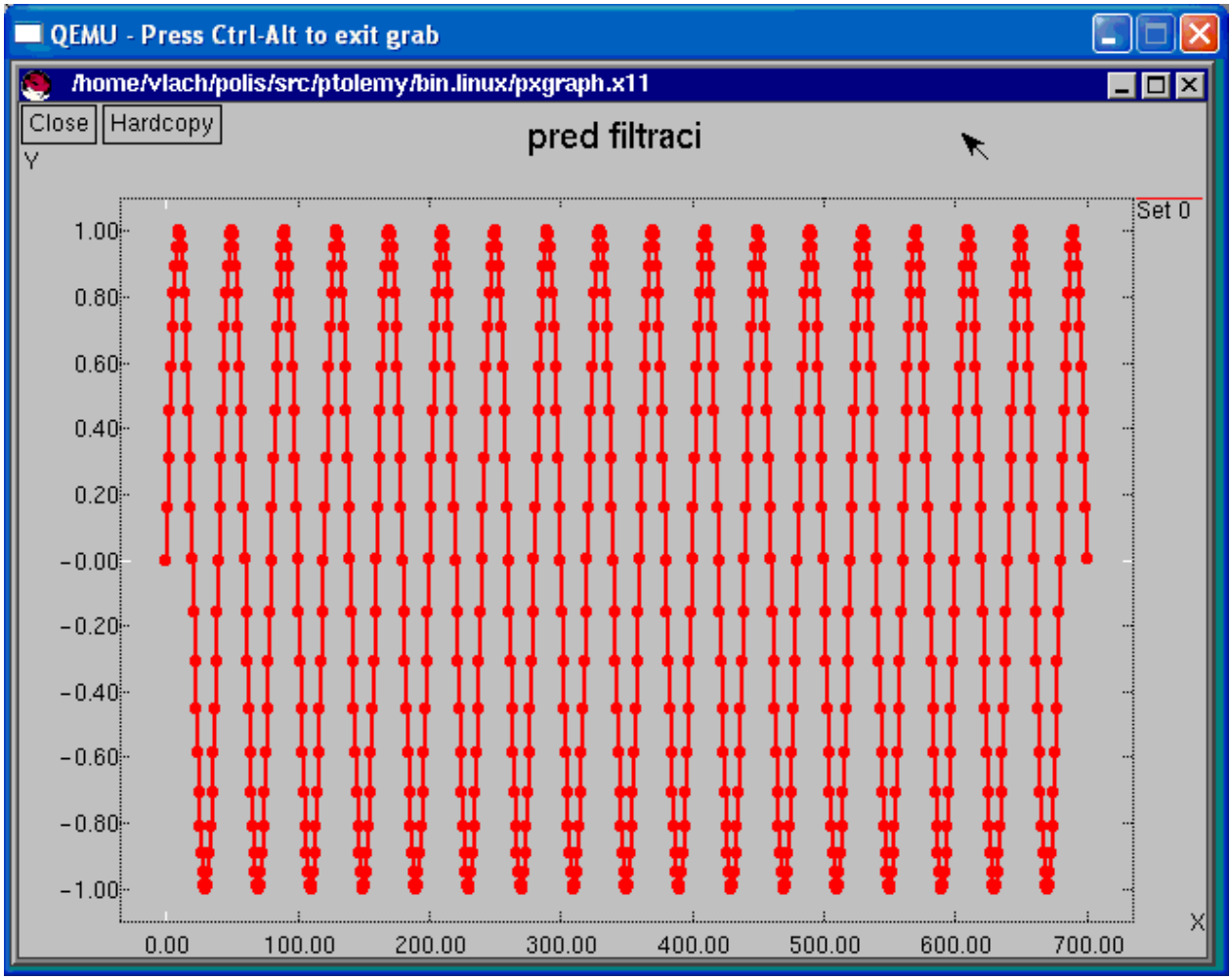


Obr.8.6 Signál 500 Hz po filtraci

Frekvence výstupního signálu FIR filtru je stejná jako u vstupního signálu, došlo však ke zmenšení amplitudy, která je nyní oproti amplitudě vstupního signálu přibližně poloviční.

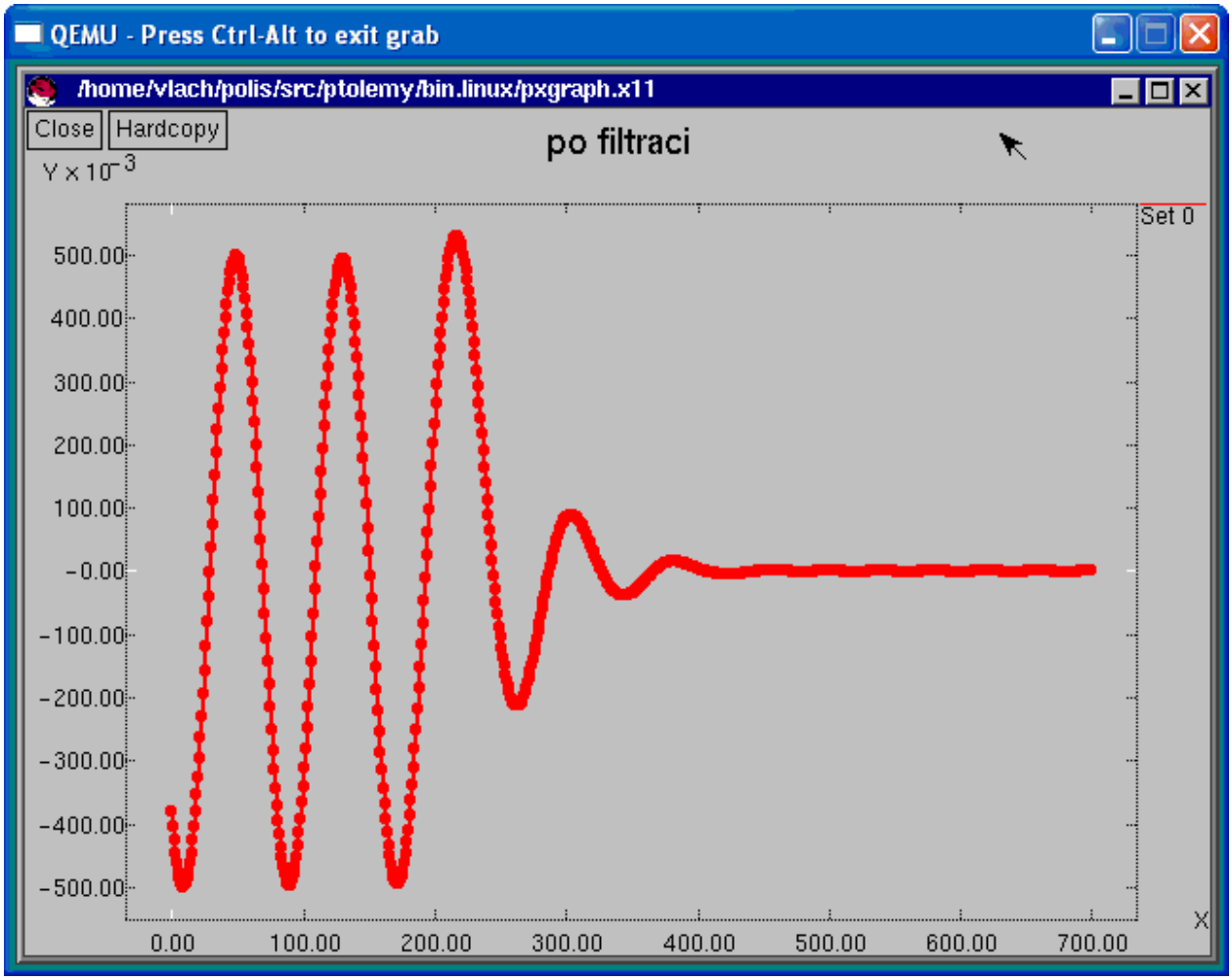
Nakonec jsem se pokusil přivést na vstup FIR filtru frekvenci 1000 Hz, která se nachází v nepropustném pásmu filtru. To znamená, že jsem parametr "frequency" galaxie "singen" domény SDF nastavil na hodnotu 2000 * PI.

Vstupní signál má ještě kratší periodu než v předchozím případě, amplituda však zůstává stále na hodnotě 1 (viz následující obrázek).



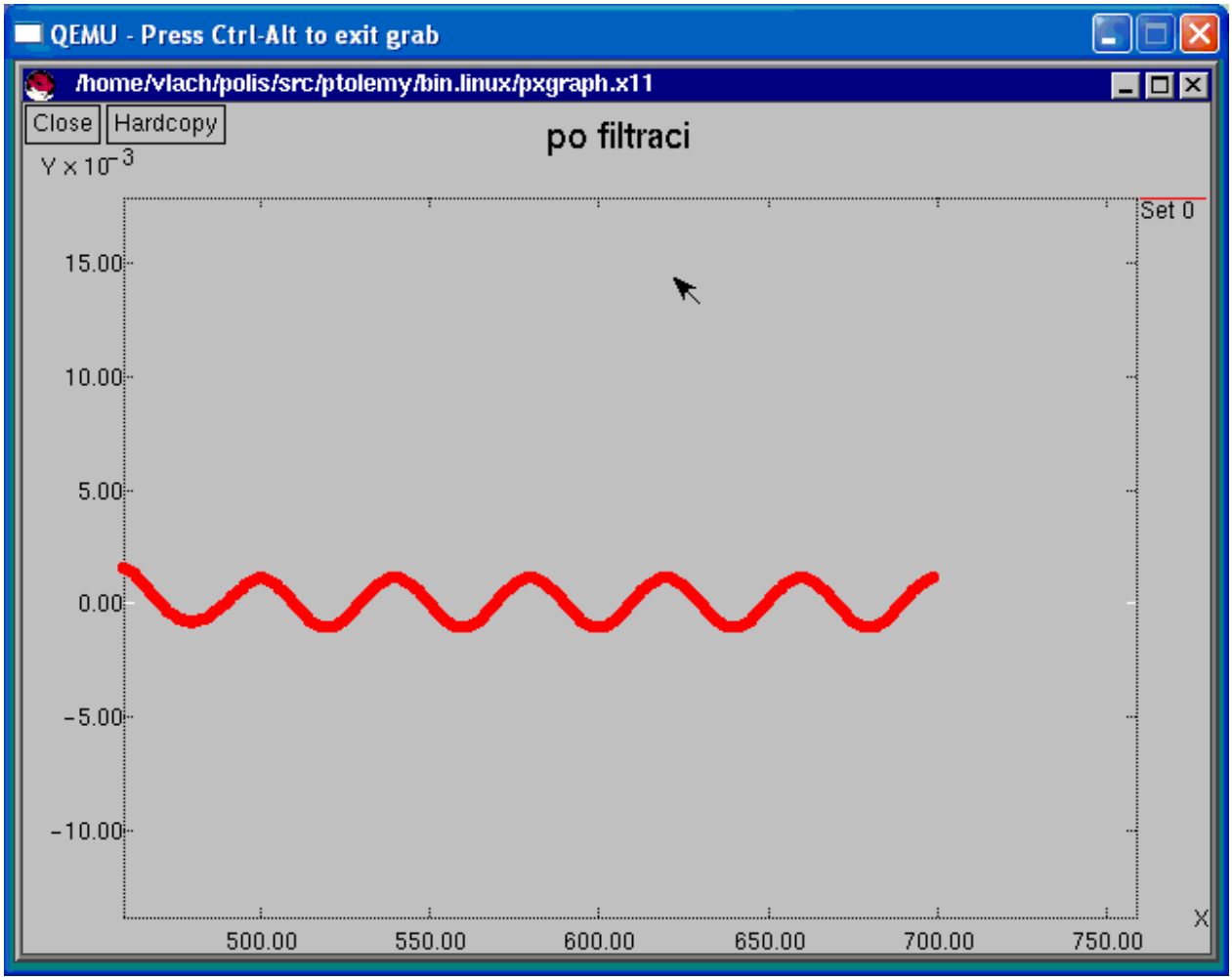
Obr. 8.7 Vstupní signál 1000 Hz

Amplituda výstupního signálu za přechodným dějem je vzhledem k amplitudě vstupního signálu výrazně utlumena. Na dalším obrázku je zachycen průběh tohoto výstupního signálu.



Obr. 8.8 Signál 1000 Hz po filtraci

Na dalším obrázku je zobrazen výřez z předcházejícího obrázku přiměřeně zvětšený tak, aby byla zřejmá amplituda výstupního signálu mimo přechodný děj.



Obr. 8.9 Signál 1000 Hz po filtraci - detail

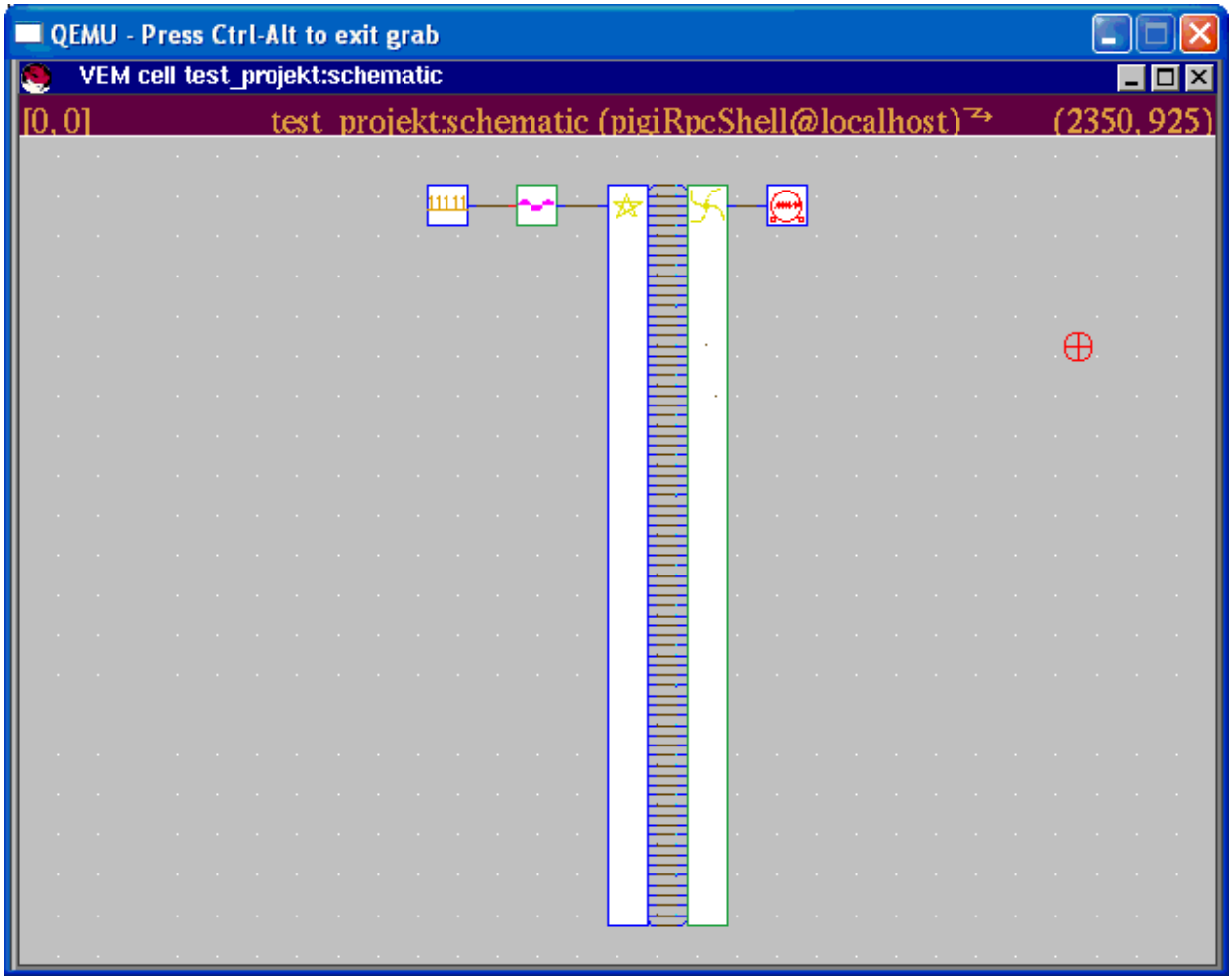
Je tedy vidět, že amplituda výstupního signálu je menší než hodnota 5.00 * 10^-3 = 0.005.

8.4 Funkční dekompozice FIR filtru

Předchozí implementace FIR filtru byla tvořena jediným CFSM, tzn. jedinou hvězdou (hvězda fir). Z tohoto důvodu bylo možné pouze implementovat celou hvězdu fir v hardwaru nebo celou hvězdu fir v softwaru.

Hvězda fir není však z hlediska své funkčnosti atomická, a lze ji tedy dekomponovat na jednodušší části, jejichž výsledná funkčnost po jejich správném propojení bude ekvivalentní původní hvězdě fir.

Funkční dekompozice je v tomto projektu provedena rozdělením obvodu FIR filtru na dvě části. Jedna z těchto částí provádí násobení (násobička) a druhá část provádí sčítání (sčítačka) (viz obr.).



Obr. 8.10 Schéma FIR filtru po dekompozici

Celé schéma je tvořeno pěti prvky. První zleva je hvězda "Clock", která generuje v předem určených časových intervalech hodinové impulsy. Druhá zleva je galaxie "Singen" domény DE, která v okamžiku každého hodinového impulsu vypočítá na základě vzorkovací frekvence a frekvence signálu pro daný okamžík hodnotu funkce sinus. Třetí zleva je hvězda "nasob", která má na vstupu sinusový signál a na svých výstupech generuje aktuální hodnoty součinů koeficientů FIR filtru a jim odpovídajících minulých vstupních hodnot. Druhý prvek zprava je galaxie "scitacka", která přijímá na svých vstupech jednotlivé součiny koeficientů FIR filtru a minulých vstupních hodnot a na výstupu generuje součet všech těchto součinů. Posledním prvkem (první zprava) je hvězda "XMgraph.input=1", která signál, jež obdrží na svém vstupu, zobrazí v grafu v nově vygenerovaném okně.

Vzhledem k tomu, že systém Ptolemy pro hvězdu, která má více než 83 výstupů, zobrazí pouze prvních 83 výstupů, rozhodl jsem se zvolit nižší řád FIR filtru, a to 74. Počet koeficientů tohoto FIR filtru je tedy 75.

Pro výpočet nových koeficientů FIR filtru jsem použil stejný program jako v předcházejícím případě - program FIRDsgn. Hodnotu vzorkovací frekvence jsem ponechal beze změny, znamenán to tedy, že její hodnota 40 000 Hz zůstala zachována. Rovněž nebyla změněna horní mezní frekvenci FIR filtru na hodnotě 500 Hz.

8.4.1 Hvězda "nasob"

Zdrojový kód hvězdy "nasob" v jazyce Esterel je umístěn v souboru nasob.strl. Obsah pomocného souboru loc_types.c zůstal téměř nezměněn. Ve funkci GET_COEF jsou pouze uvedeny jiné koeficienty FIR filtru. Místo typu Tarray501 je v tomto případě použit typ Tarray75, který se od typu Tarray501 liší pouze v počtu prvků položky array. Pole array ve struktuře typu Tarray75 má nyní pouze 75 prvků typu float.

Hvězda "nasob" má 1 vstupní signál X a 75 výstupních signálů Y0 až Y74. Pro získání požadovaného koeficientu FIR filtru slouží opět funkce GET_COEF, pro přístup k některé z minulých 75 hodnot vstupního signálu X slouží funkce GET_POLE_X. Pro zápis aktuální hodnoty vstupního signálu na začátek pole minulých 75 vstupních hodnot slouží funkce PUT_POLE_X.

Za deklaracemi těchto tří funkcí ve zdrojovém souboru nasob.strl následuje deklarace 75 pomocných proměnných o0 až o74. Do nich se budou později ukládat jednotlivé součiny. Jak vstupní signál X, tak i výstupní signály Y0 až Y74 a pomocné proměnné o0 až o74 mají svůj typ nejprve deklarován jako integer a jsou následně přetypovány v pomocném souboru types.aux na typ float.

Následuje deklarace proměnné pole_X typu Tarray75, do které jsou ukládány minulé hodnoty vstupního signálu.

Zbývající část zdrojového programu je tvořena konstruktem "every X do", tzn. že zbývající část je provedena vždy při každém dalším výskytu události na vstupním signálu X.

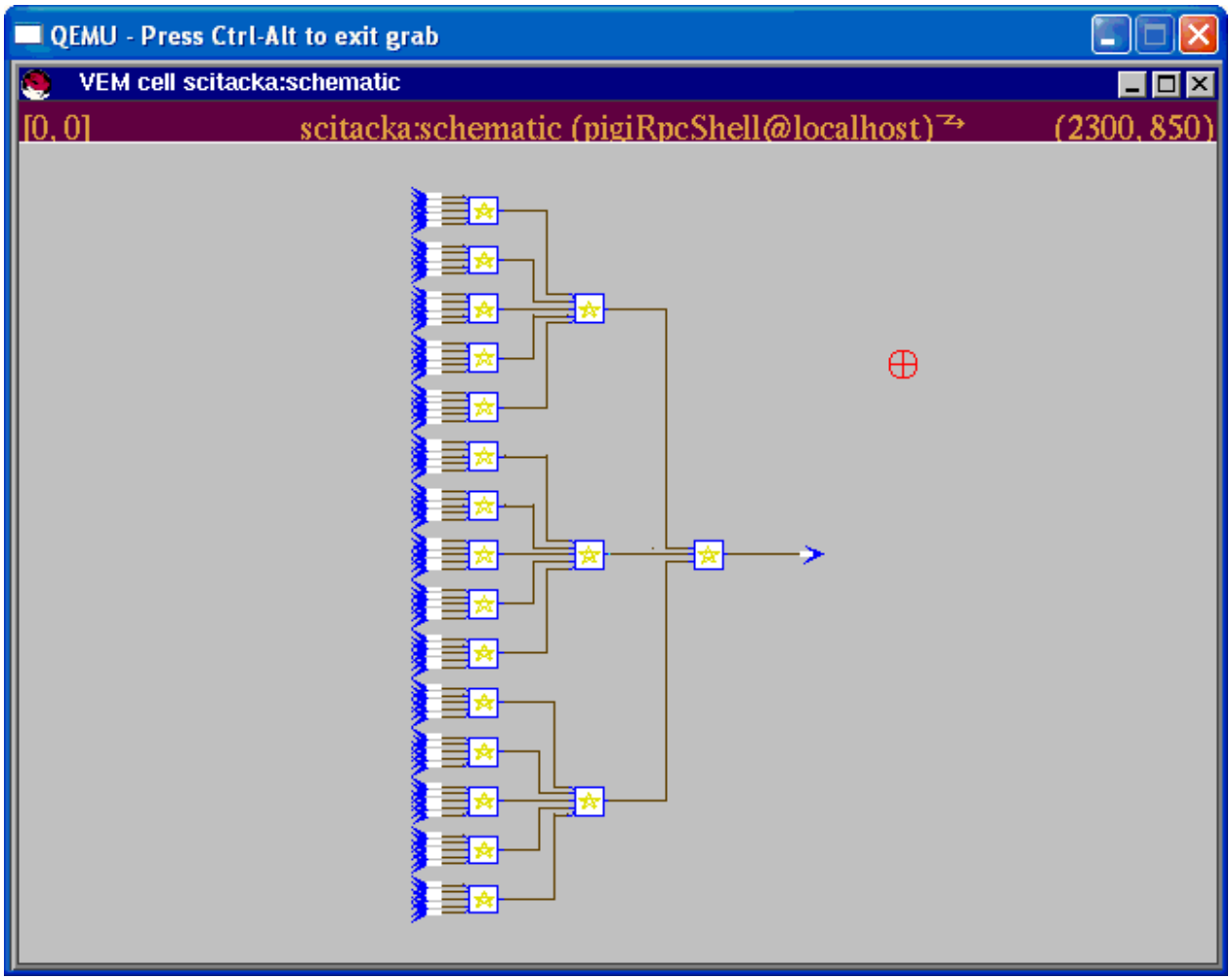
Tato zbývající část začíná přiřazením aktuálního vstupního vzorku na začátek pole array proměnné pole_X prostřednictvím funkce PUT_POLE_X. Dále následuje 75 přiřazovacích příkazů, z nichž každý obsahuje součin ve tvaru GET_COEF(i) * GET_POLE_X(i, pole_X), kde i je celočíselným prvkem z intervalu 0 včetně až 74 včetně a v žádném součinu se nesmí vyskytovat stejné "i" jako v nějakém jiném. Jsou to postupně součiny všech koeficientů a jim odpovídajících minulých hodnot vstupního signálu. Výsledek každého takového součinu je přiřazen do příslušné proměnné příkazem tvaru oi := GET_COEF(i) * GET_POLE_X(i, pole_X).

Za těmito přiřazovacími příkazy se vyskytuje příkaz "pause", a to ze stejného důvodu, který byl popsán výše u popisu modulu fir.

Následuje 75 příkazů tvaru "emitYi(oi);", pro i = 0 včetně až 74 včetně. Těmito příkazy jsou jednotlivé součiny, vypočtené dříve a uložené do proměnných oi, emitovány na jim odpovídajících výstupních signálech Yi.

8.4.2 Galaxie "scitacka"

Galaxie "scitacka" je tvořena hvězdami typu "plus5f" a hvězdou typu plus3f (viz obrázek).



Obr. 8.11 Galaxie "scitacka"

Na vstupy hvězd "plus5f" v prvním sloupci těchto hvězd zleva jsou připojeny vstupní konektory galaxie (galaxy input port) a na výstup hvězdy "plus3f" je připojen výstupní konektor galaxie (galaxy output port). Tyto konektory se nacházejí v paletě "system" a je nutné je připojit na všechny vstupy a výstupy každé galaxie.

Zdrojový kód hvězdy "plus5f" je následující:

```
module plus5f:

input XS0 : integer,
      XS1 : integer,
      XS2 : integer,
      XS3 : integer,
      XS4 : integer;

output YS : integer;
```

```
loop

  [await XS0 ||
   await XS1 ||
   await XS2 ||
   await XS3 ||
   await XS4];

   emit YS(?XS0 +
           ?XS1 +
            ?XS2 +
           ?XS3 +
           ?XS4);

end loop

end module
```

Tělo modulu je tvořeno smyčkou, ve které se čeká na výskyt události na všech pěti vstupech. Poté, co se na všech pěti vstupech vyskytla událost, je na výstupu YS emitována událost s hodnotou, danou součtem hodnot na všech vstupních signálech. Pro modul "plus5f" je v souboru types.aux použito nasledující přetypování:

```
nb XS0 31 float in plus5f
nb XS1 31 float in plus5f
nb XS2 31 float in plus5f
nb XS3 31 float in plus5f
nb XS4 31 float in plus5f

nb YS 31 float in plus5f
```

Od hvězdy "plus5f"se hvězda "plus3f" liší pouze v tom, že má místo pěti vstupů tři vstupy, jejichž součet je opět emitován jako hodnota jejího výstupního signálu. Přetypování všech vstupních signálů a výstupního signálu je provedeno stejným způsobem, jako je tomu u hvězdy "plus5f" v souboru types.aux. Zdrojový kód hvězdy "plus3f" je uveden v následujícím výpisu:

```
module plus3f:

input X0 : integer,
      X1 : integer,
      X2 : integer;

output Y : integer;

loop

  [await X0 ||
   await X1 ||
   await X2];

   emit Y(?X0 +
          ?X1 +
          ?X2);
```

end loop

end module

Na výstupu galaxie "scitacka" je tedy vždy, když se na všech jejích vstupech objeví událost, emitována událost s hodnotou, která je rovna součtu hodnot všech jejích vstupních signálů.

8.5 Hardware/software rozdělení FIR filtru

Hardware/software rozdělení projektu může být provedeno interaktivně v rámci simulačního prostředí PTOLEMY. Příkaz "make ptl", zadaný v příkazovém řádku, vytvoří automaticky pro každou hvězdu pět parametrů. Jsou to parametry "implem" typu string, "resource Name" typu string, "Clock_freq" typu float, "policy" typu string a "Priority" typu int. Mimo tyto parametry jsou pro každou hvězdu generovanou programem Polis vytvořeny ještě parametry "firename" a "overname", které mají ale pevně nastavené hodnoty, a nelze je tedy měnit.

Pro hardware/software rozdělení je rozhodující parametr "implem". Pomocí něj lze dané hvězdě přiřadit buď hardwarovou implementaci (hodnota "HW"), nebo softwarovou implementaci (hodnota "SW"). Pokud má určitá hvězda přiřazenu softwarovou implementaci, hodnota parametru "resourceName" představuje jméno procesoru, na kterém daný software poběží.

Pomocí parametru "Clock_freq" je možné nastavit frekvenci procesoru, který byl zvolen pro běh softwarových hvězd. Hodnota parametru "Clock_freq" představuje frekvenci tohoto procesoru v MHz.

Parametr "policy" slouží k výběru plánovací politiky, která bude použita pri přístupu jednotlivých hvězd k procesoru. Tento parametr se týká pouze hvězd vybraných pro softwarovou implementaci. Hvězdy, které maji parametr "implem" nastavený na hodnotu "HW", jsou prováděny vždy souběžně (concurrently).

Parametr "Priority" má význam pouze u plánovacích politik, které používají prioritu. Hvězdy, které mají stejnou hodnotu priority, jsou provedeny ve stylu plánovací politiky "RoundRobin".

Jelikož je původní hvězda fir dekomponována na dvě části, existují celkem 4 možnosti různého přiřazení implementací hvězdě "nasob" a galaxii "scitacka". Galaxie "scitacka" je sice dále dekomponována na hvězdy "plus5f" a hvězdu "plus3f", ale tyto hvězdy dědí svoji implementaci skrze hierarchické uspořádání z parametru "implem" galaxie "scitacka". Pokud tedy bude mít galaxie "scitacka" hodnotu parametru "implem" rovnu "HW", budou mít i všechny hvězdy "plus5f" a hvězda "plus3f", nacházející se v galaxii "scitacka", přiřazenu hodnotu parametru "implem" rovnu "HW". V případě, že galaxie "scitacka" bude mít parametr "implem" nastaven na "SW", budou mít všechny hvězdy v galaxii "scitacka" nastavenu softwarovou implementaci.

Aby mohl být parametr "implem" zděděn skrze hierarchii, musí být hodnota parametru "implem" u dědících hvězd nastavena na hodnotu "{implem}" (v případě parametrů typu string je nutné použít složené závorky).

8.6 Odhady hodinových cyklů a velikosti kódu programem Polis

Odhady minimálního počtu hodinových cyklů, maximálního počtu hodinových cyklů a velikosti kódu pro určitý S-graf (při simulaci v prostředí Ptolemy mají S-graf i hvězdy, jejichž parametr "implem" má hodnotu "HW", protože systém Ptolemy používá soubory v jazyce C, a bylo tedy nutné v Polisu při generování souborů pro systém Ptolemy nastavit všechny CFSM na softwarovou implementaci příkazem "set_impl -s") jsou provedeny během generování simulačních souborů pro systém Ptolemy pomocí příkazů "read_cost_param" a "print_cost".

Výběr konkrétního procesoru (neboli souboru charakterizujícího daný procesor) je možné uskutečnit pomocí maker UC a UCSUFFIX v souboru Makefile.src. Makro UC definuje rodinu mikrokontrolérů, která bude použita. Makro UCSUFFIX identifikuje sběrnicovou a paměťovou architekturu mikrokontroléru. Makro UCSUFFIX vlastně vybírá konkrétní variantu z rodiny mikrokontrolérů určené makrem UC.

Hodnoty těchto odhadů jsou zapsány během generování simulačních souborů pro systém Ptolemy do souborů ve formátu .pl. V grafickém prostředí pigi systému Ptolemy lze tyto hodnoty zobrazit zadáním příkazu :profile s ukazatelem myši nad ikonou hvězdy, jejíž odhady hodinových cyklů a velikosti kódu se mají zobrazit.

Následující přehled udává odhady hodnot všech tří výše uvedených cen (costs) pro všechny čtyři rodiny mikrokontrolérů, pro které program Polis obsahuje soubory s potřebnými parametry (minimální počet hodinových cyklů je zde označen jako "min_time", maximální počet hodinových cyklů jako "max_time" a velikost kódu jako "code_size").

Mikrokontrolér Motorola 68hc11:

- pro hvězdu "nasob":

min_time: 1639
max_time: 24432
code_size: 7378

- pro hvězdu "plus5f":

min_time: 179
max_time: 1148
code_size: 3676

- pro hvězdu "plus3f":

min_time: 159
max_time: 470
code_size: 680

Mikrokontrolér Motorola 68332:

- pro hvězdu "nasob":

min_time: 1888
max_time: 6489
code_size: 8663

- pro hvězdu "plus5f":

min_time: 136
max_time: 1099
code_size: 4095

- pro hvězdu "plus3f":

min_time: 112
max_time: 499
code_size: 745

Mikrokontrolér MIPS R3000:

- pro hvězdu "nasob":

min_time: 487
max_time: 2403
code_size: 8727

- pro hvězdu "plus5f":

min_time: 49
max_time: 206
code_size: 4672

- pro hvězdu "plus3f":

min_time: 43
max_time: 118
code_size: 882

Mikrokontrolér SPARC:

- pro hvězdu "nasob":

min_time: 779
max_time: 5853
code_size: 16

- pro hvězdu "plus5f":

min_time: 49
max_time: 285
code_size: 132

- pro hvězdu "plus3f":

min_time: 39
max_time: 129
code_size: 36

8.7 Přeložení Ptolemy netlistu do formátu SHIFT

Aby bylo možné pokračovat v následných krocích syntézy v programu Poilis, je nutné přeložit netlist ve formátu pro systém Ptolemy do formátu SHIFT.

Tento překlad lze provést ručně s využitím příkazu "ptl2shift". Další možností je zadat příkaz "make shift" v hlavním adresáři projektu. Potom je však nutné doplnit do souboru Makefile.src další makro TOPCELL, jehož hodnota musí obsahovat galaxii systému a všechny její podgalaxie. Výsledkem provedení tohoto příkazu je soubor $(TOP).shift, kde $(TOP) je hodnota makra TOP, které se nachází v souboru Makefile.src.

Zároveň se příkazem "make shift" vygeneruje také soubor $(TOP).aux, který bude obsahovat popis propojení jednotlivých prvků celého navrhovaného systému a navíc u každého prvku bude uvedena jeho implementace tak, jak byla naposled nastavena v prostředí Ptolemy. Za deklarací prvku, který byl v prostředí Ptolemy určen pro implementaci v hardwaru, je uvedeno "%impl=HW" a za deklarací prvku, kterému byla v prostředí Ptolemy přiřazena softwarová implementace, je uvedeno "%impl=SW".

Podle výše provedeného hardware/software rozdělení tedy bude u hvězdy "nasob" atribut SW a u galaxie "scitacka" bude atribut "%impl=HW", který bude skrze hierachii přenesen i na všechny hvězdy, jež se vyskytují v galaxii "scitacka". To znamená, že i všechny hvězdy "plus5f" a hvězda "plus3f" budou mít u sebe uveden atribut "%impl=HW".

9 Syntéza softwarové části

Nejprve je nutné spustit příkazový interpret programu Polis a v něm načíst celý projekt ve formátu SHIFT příkazem "read_shift $(TOP).shift". Dále je třeba zadat příkaz "partition", který vybuduje vnitřní datové struktury pro syntézu softwaru a hardwaru. Následuje příkaz "build_sg", který vytvoří graf toku řízení/dat (S-graf).

V této chvíli je zapotřebí vybrat mikrokontrolér, na kterém poběží softwarová část projektu. Program Polis nabízí výše uvedené čtyři rodiny mikrokontrolérů, z nichž všechny lze použít i v této fázi projektu. Existuje také možnost modelovat nový (další) procesor, což ale vyžaduje získat jeho cenové parametry. Cenové parametry pro cílový systém (procesor, překladač) lze zjistit například s využitím množiny benchmarkových programů, které obsahuje program Polis. Tyto programy jsou napsány v jazyce C a sestávají z asi 20 funkcí, z nichž každá je tvořena 10 až 40 příkazy.

Příkaz "sg_to_c" vygeneruje z daného S-grafu soubor s kódem v jazyce C a příkaz "gen_os" vygeneruje zdrojový soubor v jazyce C, který implementuje real-time operační systém (RTOS) pro celé softwarové rozdělení.

10 Simulace syntetizované softwarové části

Program POLIS nabízí možnost jednoduché simulace softwaru v textovém prostředí. K jeho vygenerování je potřeba buď použít příkaz "make sw" v hlavním adresáři projektu, nebo vzít syntetizovanou softwarovou část (všechny soubory softwarové části v jazyce C včetně zdrojového souboru s vygenerovaným operačním systémem os.c) a přeložit všechny tyto soubory překladačem jazyka C pomocí příkazů:

```
cc -g -c -DESTEREL -DUNIX -I$POLIS/polis_lib/os *.c
cc -g -o projekt *.o
```

Příznak "-DUNIX" vybírá pro kompilaci tu část operačního systému (tj. souboru os.c), která nahrazuje fyzické vstupy a výstupy textovým simulačním rozhraním.

Volitelný příznak "-DESTEREL" má za následek, že simulátor nevytiskne hodnoty výstupních singálů okamžitě, nýbrž dále plánuje jednotlivé CSFM až do doby, kdy systém dosáhne vnitřní stability. Teprve potom se na obrazovku vytisknou výstupní signály se svými hodnotami.

Jméno výsledného spustitelného souboru jsem zvolil shodně s hodnotou makra TOP v souboru Makefile.src, tedy "projekt".

Po spuštění tohoto souboru v příkazovém řádku příkazem ./projekt se na obrazovce objeví výzva (prompt) ve tvaru "os sim>". Do něj je možné zadávat jména vstupních signálů s jejich hodnotami uvedenými v závorkách za nimi. Jednotlivé signály se zadávají tak, aby byly navzájem odděleny mezerou a celý seznam těchto singálů je ukončen středníkem. Po stisknutí klávesy Enter se na obrazovce objeví jména výstupních signálů, které mají u sebe v závorkách uvedeny hodnoty, jež byly změněny na základě zadaných hodnot vstupních signálů.

Abych bylo možné jednoduše ověřit správnou funkci vygenerovaného souboru "z_nasob_0.c", který by měl provádět funkci specifikovanou modulem "nasob", implemenoval jsem pro tento účel v softwaru též zbývající část projektu, která je určena pro implementaci v hardwaru. Takto vygenerovaný simulátor má jeden vstup, pojmenovaný jako "vstup", na který postupně přicházejí vzorky vstupního signálu, a jeden výstup, pojmenovaný "výstup", na kterém jsou generovány vzorky přefiltrovaného signálu.

Pro snadnější zadávání vstupních vzorků tak, aby nebylo nutné zadávat postupně ručně hodnoty vstupního signálu do simulátoru v interaktivním režimu, jsem si vytvořil pomocný program sin.cpp v jazyce C++, který do souboru "sin_vstup.txt" vygeneruje prvních 200 vzorků diskrétního sinusového signálu, a to jednou s hodnotou frekvence 200 Hz a jednou s hodnotou frekvence 1000 Hz vzhledem k vzorkovací frekvenci 40 000 Hz.

Tento soubor se přivede na vstup simulátoru tak, že se nejprve simulátor spustí a v jeho příkazovém řádku (s promptem "os sim>") se zadá příkaz "_newfile <název souboru s hodnotami vstupních signálů (v tomto případě tedy soubor "sin_vstup.txt")>".

Pro lepší možnost zpracování výsledků jsem simulátor spustil s přesměrováním jeho výstupu do souboru. V případě frekvence vstupního sinusového singálu 200 Hz jde o soubor "vystup-sin200Hz" a v případě frekvence 1000 Hz o soubor "vystup-sin1000Hz".

V souboru "vystup-sin200Hz" je možné pozorovat, že amplituda dosahuje hodnoty 0.928138, tedy vstupní signál prochází FIR filtrem nezměněn, což je v souladu s předpokládaným chováním, jelikož frekvence 200 Hz se nachází v propustném pásmu filtru.

V souboru "vystup-sin1000Hz" má amplituda výstupu FIR filtru "vystup" hodnotu 0.120429 (ani v tomto případě neuvažuji hodnoty výstupu během počátečního přechodového děje). Došlo tedy k výraznému utlumení vstupního signálu, které opět odpovídá předpokládanému chování, jelikož frekvence 1000 Hz se nachází v nepropustném pásmu tohoto FIR filtru.

11 Cena softwarové implementace FIR filtru

Podobně, jako byly v jedné z předchozích částí zjišťovány ceny jednotlivých hvězd v rámci celého návrhu, lze stejný postup aplikovat na celý návrh za předpokladu, že všechny prvky FIR filtru budou mít přiřazenu softwarovou implementaci.

Postup je téměř stejný jako v případě ohodnocování jednotlivých hvězd. Pouze místo souboru ve formátu SHIFT pro hvězdu se v příkazu "read_shift" v interpretu Polisu použije soubor SHIFT popisující celý návrh. Je to tentý soubor, který byl získán po skončení fáze kosimulace v prostředí Ptolemy příkazem "make shift". To znamená, že pokud makro TOP v souboru Makefile.src má hodnotu projekt, potom se tento soubor jmenuje projekt.shift a příslušný pomocný soubor projekt.aux.

V tomto případě se po zadání příkazu "print_cost -s" v interpretu Polisu vypíší jednak všechny tři cenové parametry pro všechny hvězdy obsažené v celém návrhu a jednak se pod to vypíší všechny tři cenové parametry pro celou softwarovou implementaci.

V následujícím výpisu jsou uvedeny odhady minimálního počtu hodinových cyklů (min_time), maximálního počtu hodinových cyklů (max_time) a velikosti kódu (code_size) pro celý FIR filtr. Tyto odhady jsou opět provedeny pro všechny čtyři procesorové rodiny, jejichž charakteristiky jsou dostupné v programu Polis.

Mikrokontrolér Motorola 68hc11:

min_time: 1977
max_time: 26050
code_size: 11734

Mikrokontrolér Motorola 68332:

min_time: 2136
max_time: 8087
code_size: 13503

Mikrokontrolér MIPS R3000:

min_time: 579
max:time: 2727
code_size: 14281

Mikrokontrolér SPARC:

min_time: 867
max_time: 6267
code_size: 184

Zdá se, že pro všechny sledované procesory platí, že v případě všech tří cenových parametrů je cena celého návrhu dána součtem cen jednotlivých hvězd.

12 Simulace a syntéza do VHDL

12.1 Simulace ve VHDL

Program Polis je schopen pro projekt načtený ve formátu SHIFT vygenerovat také kód v jazyce VHDL.

Proces vytváření simulačního modelu pro simulaci ve VHDL je ze začátku shodný s vytvářením simulačních souborů pro systém Ptolemy. Moduly se zapíší v jazyce Esterel a přeloží se do formátu SHIFT. Vytvoří se pomocný soubor s netlistem a vygeneruje se soubor SHIFT pro celý návrh. Ten se načte příkazem "read_shift". Poté se provede rozdělení příkazem "partition" a vybudují se S-grafy příkazem "build_sg". Namísto příkazů "sg_to_c" a "write_pl" v příkazovém interpretu Polisu se však použije příkaz sg_to_vhdl. Je rovněž možné místo toho použít příkaz "make beh_vhdl", který se zadá v příkazovém řádku v hlavním adresáři projektu. Tento příkaz vygeneruje soubor $(TOP).vhd, který může být načten do komerčního simulátoru.

Pro softwarové CFSM napodobuje kód ve VHDL strukturu S-grafu. Tento kód je opatřen poznámkami, ve kterých jsou odhadnuté informace o zpožděních jednotlivých částí kódu, stejně jako je tomu v případě kódu v jazyce C pro simulaci v systému Ptolemy.

Pro CFSM určené pro hardware se vygeneruje RTL (register transfer level) kód z jim odpovídajících S-grafů.

Tato technika zachází stejným způsobem se softwarovými i s hardwarovými CFSM, podobně jako je tomu v případě simulace v systému Ptolemy. RTL kód, který popisuje cestu v S-grafu, potom odpovídá přechodu CFSM a potřebuje právě jeden hodinový cyklus.

Hardwarové CSFM, které jsou syntetizované v systému Polis mohou být transformovány do VHDL čtyřmi způsoby.

Prvním způsobem je použití S-grafu, avšak bez časových informací. Aby se použil tento způsob, je nutné přiřadit hardwarovým CFSM softwarovou implementaci pomocí příkazu "set_impl -s", vybudovat S-graf a použít příkaz sg_to_vhdl bez volby "-D", aby se do výsledného VHDL souboru nepřidávaly instrukce pro čítání hodinových cyklů. Tento postup generuje RTL VHDL kód pro určité hardwarové CFSM. Příkaz sg_to_vhdl může být tedy použit pro celosoftwarovou implementaci s volbou "-D" a pro celohardwarovou implementaci bez volby "-D". Pokud se použije volba "-A", vygeneruje se asynchronní Mealyho implementace, pokud se volba "-A" nepoužije, je vygenerována synchronní Mealyho implementace. V případě asynchronní Mealyho implementace nemají výstupy registry, zatímco v případě synchronní Mealyho implementace jsou výstupy registry opatřeny.

Druhou možností je použití příkazu "hw_to_vhdl" pro vygenerování RTL VHDL kódu pro hardwarové rozdělení. Musí být také použit příkaz "sg_to_vhdl", aby byla vygenerována entita a architektura pro každé softwarové CFSM netlist na nejvyšší úrovni (top-level) pro softwarové rozdělení. Tento postup je potřeba vybrat pro smíšenou hardware/software kosimulaci ve VHDL. Implementace může být změněna dynamicky a VHDL může být regenerován, aby se ohodnotilo nové rozdělení pomocí souběžné simulace.

Třetí možností je použít namapovaný soubor ve formátu blif z fáze syntézy hardwaru v programu

Polis a přeložit tento soubor do strukturálního kódu v jazyce VHDL tak, že se použije nástroj blif2vst, který je součástí distribuce Polisu. Tento přístup generuje hardwarové CFSM na úrovni hradel (gate level) místo úrovně RTL a je podporován standardním mechanismem překladu pomocí souborů makefile tak, že lze použít příkaz "make map-vhdl" v příkazovém řádku v hlavním adresáři projektu.

Poslední, čtvrtou, možností je použití skriptu "shift2vhdl".

Základní myšlenkou ve schématu pro generování VHDL kódu je, že S-graf je interpretován jako konečný automat (FSM), ve kterém existuje pro každý uzel S-grafu jeden stav. Provedení S-grafu od začátku (uzel BEGIN) do konce (uzel END) se pak stane seřazeným průchodem sekvencí stavů tohoto konečného automatu. Tento konečný automat lze chápat jako sekvenční implementaci přechodové funkce příslušného CFSM.

12.2 Syntéza VHDL kódu

Pro syntézu VHDL kódu programem Polis je určen skript "shift2vhdl", který z návrhu projektu ve formátu SHIFT vygeneruje RTL VHDL kód. Na rozdíl od jiných příkazů nebere v úvahu aktuální hardware/software rozdělení, ale aplikuje se na všechny CFSM. VHDL kód vygenerovaný tímto skriptem obsahuje jednu entitu pro hlavní síť komunikující procesy (communicating processes) pro jednotlivá CSFM.

Tento VHDL kód vyžaduje, aby byly do knihovny WORK zkompilovány další dva soubory, které jsou součástí distribuce Polisu. Jde o soubory numeric_bit.vhd a routines.vhd, které obsahují package numeric_bit a package uC_routines_synth_pkg.

Package uC_routines_synth_pkg obsahuje mnoho aritmetických a logických funkcí, které jsou volány z hlavního vygenerovaného VHDL souboru a mnoho těchto funkcí je zde uvedeno vícekrát pro různé typy parametrů a pro různé návratové typy těchto funkcí. Přesto zde nejsou vyčerpány všechny možné kombinace.

12.3 Uživatelské funkce ve VHDL

Uživatelské funkce, které jsou deklarovány ve zdrojových souborech v jazyce Esterel a které jsou definovány v pomocném souboru loc_types.c, se při vytváření simulačních souborů pro systém Ptolemy přeloží a jsou také začleněny do systému Ptolemy.

V případě překladu návrhu do jazyka VHDL se však přeloží pouze ta část návrhu, která vznikla pouze ze zdrojových souborů v jazyce Esterel. Pokud návrh obsahuje pomocné funkce definované v jazyce C, je nutné tyto funkce znovu napsat v jazyce VHDL a přidat je k vygenerovaným souborům VHDL. Vygenerovaný kód VHDL však obsahuje volání těchto funkcí tak, jak je tomu ve zdrojových souborech v jazyce Esterel.

Stejná situace panuje i u přetypování vstupních a výstupních signálů a proměnných v pomocném souboru .aux např. z typu integer na typ float. Je potřeba tyto proměnné ručně přetypovat ve výsledném souboru v jazyce VHDL.

12.4 Uživatelské funkce pro FIR filtr

Návrh FIR filtru obsahuje tři uživatelské funkce definované v souboru loc_types.c

Vytvořil jsem vlastní soubor funkce.vhd, který obsahuje package FUNKCE, ve kterém je na začátku uvedena definice typu Tarray501, který zde však není ve formě struktury, ale jedná se přímo o pole 501 položek typu real. Následuje deklarace funkce GET_COEF, která přebírá jeden reálný parametr a vrací hodnotu typu real.

Za tímto se již vyskytuje tělo balíčku (package) FUNKCE, které má v sobě definici funkce GET_COEF, mající zde v podstatě stejné tělo jako v případě souboru loc_types.c. Tato funkce používá dvě lokální proměnné, proměnnou array_pom typu Tarray501 a proměnnou i typu integer. Dalšími příkazy jsou přiřazovací příkazy, které do jednotlivých položek pole array_pom ukládají hodnoty příslušných koeficientů FIR filtru. Proměnná "i" slouží pouze k přetypování hodnoty indexu na celé číslo, které se již použije pro výběr požadovaného koeficientu. Tím package FUNKCE končí.

Další funkce, funkci PUT_POLE_X a funkci GET_POLE_X jsem definoval přímo v hlavním vygenerovaném souboru s VHDL kódem, v souboru z_projekt-SW.vhd.

Pro ukládání minulých 501 hodnot vstupního signálu jsem vytvořil proměnnou poleX. Tuto proměnnou jsem deklaroval jako sdílenou (shared), aby bylo možné k ní přistupovat i uvnitř těla obou výše zmíněných funkcí. Ze stejného důvodu jsou obě funkce deklarovány jako nečisté (impure).

Pro otestování funkčnosti vygenerovaného souboru v jazyce VHDL jsem vytvořil soubor test.vhd, ve kterém instantizuji komponentu z_projekt a na její vstup přivádím sinusový signál. Abych mohl použít funkci SIN ve zdrojovém kódu VHDL, přidal jsem do knihovny WORK package MATH_REAL, který se nachází v souboru mathpack.vhd a který funkci SIN obsahuje. Hodnotě signálu v_vstup_test_S, který je použit pro nesení hodnot vstupního signálu, přiřadím hodnotu funkce SIN(x). Argumentem funkce SIN je proměnná x typu real, která je na počátku inicializovaná na hodnotu 0.0. Při každém výskytu události na singálu e_vstup_test_S, který představuje událost příchodu vzorku vstupního signálu do FIR filtru, se hodnota proměnné x aktualizuje podle stejného vzorce, podle jakého se zvyšovala hodnota výstupu hvězdy "Ramp" v simulaci FIR filtru v ptostředí Ptolemy, tj, podle vzorce

x := x + (2 * PI * frekvence_vstupního signálu / vzorkovací_frekvence).

Úhlové frekvence zde není nutné uvažovat, neboť by se jednalo pouze o rozšíření zlomku ve výše uvedeném výrazu hodnotou 2 * PI.

13 Závěr

Tento diplomový projekt navazuje na můj ročníkový projekt (PI2), který se zabýval teorií souběžného návrhu technického a programového vybavení vestavěných systémů. Uvedený ročníkový projekt obsahoval i stručný popis programu Polis. Toto téma dále rozvíjí předmětný diplomový projekt, který se věnuje i souběžné simulaci technického a programového vybavení, simulačnímu systému Ptolemy a podrobněji i programu Polis.

Zpracovaný projekt je příspěvkem k řešení problematiky souběžného návrhu technického a programového vybavení vestavěných systémů s využitím programových nástrojů vyvíjených na University of California, Berkeley (USA).

Na tento projekt lze navázat implementací hardwarových CFSM v FPGA. FIR filtr navrhovaný v tomto projektu lze uplatnit jako filtr zvukového signálu tak, že se jeho určité části začlení do již hotových programů pro získávání dat z audio vstupu (s tím, že přefiltrovaný zvukový singál bude mít zeslabené frekvence, které jsou vyšší než je hodnota horní mezní frekvence (cut-off) tohoto FIR filtru) a zasílání vzorků přefiltrovaného zvukového signálu na audio výstup.

Literatura

[1] Jialong, He *www.bigfoot.com/~Jialong_He*

[2] Sanjaya Kumar (Honeywell Technology Center), James H. Aylor, Barry W. Johnson, WM. Wulf (University of Virginia): The codesign of embedded systems: A unified hardware/software representation. Kluwer Academic Publishers, 1996, ISBN 0-7923-9636-7

[3] http://embedded.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

[4] http://ptolemy.eecs.berkeley.edu/

Seznam příloh